Contribution to solving the combinatorial optimization problems : case of the problem of K-clusters in a bipartite graph and the quadratic knapsack problem

Najat Hameed Qasim Al-Iedani

Thèse de Doctorat

Spécialité Informatique

présentée à ***l'Ecole Doctorale en Sciences Technologie et Santé (ED 585)***

de l'Université de Picardie Jules Verne

par

Najat Hameed Qasim Al-Iedani

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne

Contribution à la résolution des problèmes d'optimisation combinatoire: cas du problème des k-clusters dans un graphe biparti et du problème de sac à dos quadratique

Soutenue le 14/11/2017 après avis des rapporteurs, devant le jury d'examen :

M. Jérôme BOSCHE,	**Maître de Conférences**	**Examinateur**
M. Haoxun CHEN,	**Professeur**	**Rapporteur**
M. Mhand HIFI,	**Professeur**	**Directeur**
M. Imed KACEM,	**Professeur**	**Président**
M. Anass NAGIH,	**Professeur**	**Rapporteur**
M. Toufik SAADI ,	**Maître de Conférences**	**Directeur**

Preface

This thesis has been performed at the unite of researches (EPROAD EA 4669)[1] in the University of Picardie Jules Verne(UPJV), since the December 2014 to October 2017. The work has been supervised by professor Mhand Hifi[2] and M.Toufik Saadi[3].

Our work deals with the combinatorial optimization problems that we have been presented within a general introduction, a state of the art and two separate papers.

First: considering a problem within the K-clusters in the bipartite graph, (Neighborhood Search-based Heuristic for the K-Clustering Minimum Biclique Completion Problem). This part of the thesis has been published in an international conference with proceedings, the 3rd IEEE, International Conference on Control, Decision and Information Technologies (CoDIT-2016), (see Hifi *et al.* [54]). And was presented during the 17th and 18th Conference of the French Society for Operational Research and Decision, ROADEF-2016 [55], ROADEF-2017 [56].

Second: considering a problem within the quadratic knapsack problem, (Reactive Search for the Quadratic Knapsack Problem). This part of the thesis we have been presented in chapter 6 that has been published in an international conference with proceedings, the 4th IEEE, International Conference on Control, Decision and Information Technologies (CoDIT-2017), (see Hifi *et al.* [57]).

Acknowledgment

I would like to thank Professor Mhand Hifi and Doctor Toufik Saadi, for them advice and support to advances in this work.

I would like to express my gratitude to the ministry of higher education and for the martyrs foundation of Iraq, for supported by research grants that nancing my study. Finally, i am very grateful to my family, especially my late parents, my martyr brother and to my husband who has supported me and bore with me during these last years and unconditional support and patience over the years. Thanks for their motivating me the rest of the time and offering comfort when necessary.

[1] Eco-Procédés, Optimization et Aide à la Décision

[2] Professor at the University of Picardie Jules Verne

[3] Associate professor at the University of Picardie Jules Verne

Résumé

Titre: Contribution à la résolution des problèmes d'optimisation combinatoire: cas du problème des K-clusters dans un graphe biparti et du problème de sac à dos quadratique.

Les problèmes d'optimisation combinatoire sont d'un grand intérêt à la fois pour le monde scientifique et le monde industriel. Les enjeux scientifiques, économiques, environnementaux et sociaux sont très nombreux et très importants. C'est pour cela que la communauté scientifique mondiale recherche depuis longtemps des méthodes de modélisation, de simplification et de résolution de ces problèmes.

Parmi les problèmes combinatoires les plus connus se trouvent les problèmes de sac à dos et les problèmes liés aux décompositions des graphes. Nous nous sommes intéressés dans cette thèse à deux problèmes importants :

- Le problème de regroupement dans un graphe biparti;
- Le problème de sac à dos quadratique;

Le problème de regroupement dans un graphe biparti a de nombreuses applications dans le domaine des télécommunications. Il a également un grand intérêt théorique dans la modélisation, la décomposition et la résolution de plusieurs autres problèmes combinatoires. Le problème de sac à dos quadratique a un large champ d'applications théoriques et pratiques dans nombreux domaines. Il est très utile dans la modélisation et la résolution dans un contexte de gestion des exclusions par exemple.

Ces deux problèmes sont hautement combinatoires et sont très difficiles à résoudre d'une manière optimale d'un point de vue informatique. La résolution de ce type de problèmes peut se faire de deux manières :

- La résolution optimale, dite également exacte, qui s'appuie sur des modélisations et des méthodes mathématiques puissantes dont l'objectif est d'identifier une solution optimale du problème traité.
- La résolution approchée, qui s'appuie principalement sur des algorithmes capables d'approcher la solution optimale du problème traité mais sans garantir l'optimalité du résultat.

Les méthodes approchées sont les plus utilisées par les informaticiens dans la résolution des problèmes issus de l'industrie et des services car ces méthodes permettent de résoudre des problèmes de grande taille et de répondre aux exigences fonctionnelles des donneurs d'ordres.

Il existe aussi des méthodes de résolution hydrides qui peuvent combiner plusieurs méthodes de résolution approchées ou exactes et qui utilisent généralement des techniques de décomposition du problème initial pour permettre l'hybridation. C'est dans ce sens que s'oriente cette thèse.

Nous proposons dans cette thèse deux méthodes de résolution hydriques :

- Une première méthode hydride pour le problème de regroupement dans un graphe biparti qui combine une recherche par voisinage et un algorithme approché complémentaire.
- Une deuxième méthode hybride pour le problème du sac à dos quadratique qui combine une recherche par voisinage et une méthode de réduction/fixation des variables.

`Mots clés`: Biclique, Cluster, Optimisation Combinatoire, Greedy, Heuristique, K-CmBCP, Recherche Locale, Recherche Opérationnelle, QKP, Réduction, Recherche réactive, sac à dos quadratique.

Abstract

Title: Contribution to solving the combinatorial optimization problems: case of the problem of K-clusters in a bipartite graph and the quadratic knapsack problem.

Since long time, the scientific world has sought for modeling, simplification and resolution of combinatorial optimization problems, because of these problems are most interest for the scientific and the industrial world and for the fields of operational research and computer science.

The objective of this thesis is to solve the difficult combinatorial optimization problems using approximate resolution methods. And, we were interested on two important problems that find several significant applications in real world.

The first part of the thesis is devoted to the K-clusters in a bipartite graph that has been applied in the field of telecommunication. The second part of the thesis addresses to the quadratic knapsack problem that can be used to accommodate a wide range of practical applications in numerous fields. On the other hand, these problems are highly combinatorial and difficult to solve from computational perspective.

The K-clustering minimum bi-clique completion problem (K - CmBCP) was presented in the latest date and it is very significant in real world and it has been applied to several real applications such as aggregation of multicast sessions. Since telecommunication network cannot manage many multicast sessions at the same time, it is hence necessary to group the sessions into a limited number of clusters.

We note that, the hybrid resolution methods can combine several approximate resolution methods or optimal resolution and approximate resolution and which generally use decomposition techniques of the initial problem to allow hybridation. In this thesis, we propose two hybrid resolution methods: A first hybrid method for the problem of K-clusters in a bipartite graph that combines a neighborhood search and a complementary algorithm. A second hybrid method for the quadratic knapsack problem which combines a large neighborhood search with a variable reduction / fixing method. The proposed algorithm is capable of solving the small, large and very large size instances of the QKP that cannot be solved by Cplex solver or by other methods.

`Keywords`: Biclique, bipartite, Cluster, Combinatorial Optimization, Greedy, Heuristics, K-CmBCP, local search, Operations Research, QKP, Reduction, reactive search, quadratic knapsack .

Contents

List of Figures

List of Tables

Chapter 1

Chapter 1

Résumé de thèse en Français

1.1 Introduction

La communauté scientifique mondiale recherche depuis longtemps des méthodes de modélisation, de simplification et de résolution des problèmes d'optimisation combinatoire car les enjeux scientifiques, économiques, environnementaux et sociaux de ces problèmes sont très importants.

Suivant l'objectif de la résolution et la taille des problèmes à traiter, ce type de problèmes peut être traité par deux familles de méthodes: (i) résoudre ces problèmes de façon optimale, en utilisant des méthodes exactes et (ii) chercher les solutions d'approximation en utilisant les méthodes heuristiques.
Les méthodes heuristiques ont reçu plus d'attention de la part des chercheurs. Néanmoins, nous notons qu'il existe d'autres méthodes de résolution, appelées méthodes hybrides qui combinent deux ou plusieurs méthodes qui peuvent être exactes ou approchées.
Nous nous sommes intéressés dans cette thèse à deux problèmes importants de l'optimisation combinatoires bien connus en recherche opérationnelle et qui trouvent plusieurs applications dans le domaine des télécommunications et la logistique.
La première partie de la thèse est consacrée au problème de k-clusters dans un graphe bipartite. La deuxième partie traite le problème de sac à dos quadratique. Les deux problèmes sont difficiles à résoudre d'un point de vue informatique et sont très utiles dans de nombreux domaines scientifiques et applicatifs.

Pour ces deux problèmes, nous proposons une méthodologie de résolution hybride qui combine des heuristiques et les méthodes de recherche par voisinage. Ce chapitre fournit une vue panoramique du contenu de ce mémoire de thèse et montre quelques domaines d'application des deux problèmes traités.

Organisation de la thèse

Cette thèse comporte deux parties:

- La première partie est composée de deux chapitres. Elle présente nos travaux et résultats sur le problème de K-clusters dans un graphe biparti.

- La deuxième partie comporte trois chapitres. Elle présente nos travaux et résultats sur problème du sac à dos quadratique.

Plus précisément, cette thèse est organisée comme suit. La première partie présente un état de l'art du problème de K-clusters dans un graphe biparti, qui consiste à trouver un nombre K de *clusters* minimisant le nombre d'arrêtes à ajouter pour obtenir K bicliques. Ce problème est plus connu sous le nom de K-Clustering minimum Biclique Completion Problem (noté: K-CmBCP) et il est NP-difficile, (vois [81]). Dans ce chapitre, nous commençons d'abord par rappeler quelques notions de la théorie des graphes afin d'établir la notation qui sera utilisée tout au long de cette partie de la thèse. Ensuite, nous présentons le problème. Nous présentons, ensuite, les méthodes de résolution utilisées dans la littérature, ainsi que quelques exemples d'applications et les principaux modèles mathématiques. Enfin, nous présentons la méthode de résolution que nous proposons pour ce problème. Le deuxième chapitre présente une méthode hybride qui combine les méthodes heuristiques et les méthodes de recherche par voisinage pour résoudre approximativement le problème du K-clusters dans un graphe biparti.

La deuxième partie de cette thèse comporte trois chapitres et elle concerne l'étude du problème du sac à dos quadratique. Dans le chapitre 5, nous abordons les méthodes de résolution des problèmes combinatoires qui peuvent être classées en deux catégories: Les méthodes exactes qui garantissent l'optimalité et les méthodes approchées (heuristiques et méta-heuristiques) qui donnent des solutions approchées.

Dans le chapitre 6, nous présentons le premier algorithme réactive pour résoudre le problème du sac à dos quadratique. Cet algorithme comporte deux méthodes: une première méthode gloutonne basée sur les principes de résolution de problème de sac-à-dos binaire (noté $0-1$ QKP), et une deuxième méthode qui comporte, elle aussi, deux étapes complémentaires: une première étape de destruction et une deuxième étape de reconstruction (ou de ré-optimisation). La méthode de résolution proposée peut être vue comme une méthode de recherche aléatoire par voisinage. En effet, l'objectif principal de ce travail est de présenter un algorithme efficace qui permet de produire des solutions de bonne qualité en un temps d'exécution raisonnable. Ainsi, l'étape de ré-optimisation est composée de deux procédures: une première procédure qui fournit une solution réalisable et une deuxième procédure qui s'intéresse à l'amélioration de la qualité de la solution. L'étape de destruction effectue une suppression aléatoire d'un nombre d'éléments appartenant à la solution dans le but de construire une nouvelle solution dans le voisinage. Notons que cette étape est considérée comme la plus importante car elle permet d'élargir l'espace de recherche tout en gardant certains éléments de la solution courante. En plus, cette étape a pour but d'éviter une série d'optima-locaux lors de la recherche. En effet, la solution produite par l'étape de destruction est traitée comme un problème réduit qui peut être optimisé par diverses approches.

Dans le chapitre 7, nous présentons le deuxième algorithme réactive pour résoudre le problème du sac à dos quadratique. Cette méthode est capable de guider le processus de recherche vers de nouveaux espaces de recherche et combine

le principe d'une recherche par voisinage et d'une stratégie de réduction des variables. La stratégie de réduction des variables est utilisée pour construire une série de sous-problèmes. Ensuite, la méthode utilise une optimisation exacte pour résoudre d'une manière exacte les sous problèmes identifiés.

Finalement, le chapitre 8 donne une conclusion générale, dans laquelle nous présentons les résultats obtenus ainsi qu'une orientation et perspectives pour des travaux futurs dans ce domaine.

1.2 Le problème de K-clusters dans un graphe biparti

Le problème de K-clusters dans un graphe biparti, qui est un problème d'optimisation combinatoire peu étudié dans la littérature. Dans ce problème, l'objectif consiste à répartir un nombre de services entre deux groupes (clusters), où chaque cluster peut contenir des services et des clients. Chacun des services doit satisfaire au moins un client. Ce problème peut être représenté sous forme d'un graphe biparti, où les services représentent les sommets du côté gauche du graphe et les clients représentent les sommets du côté droit du graphe biparti. Ainsi, afin de contribuer à la résolution de ce problème, nous avons proposé une méthode de résolution approchée déterminant le meilleur partitionnement des services permettant de satisfaire les clients à moindre coût [51].

Comme décrit dans Gualandi et al [80], le K-CmBCP a de nombreuses applications réelles comme le partitionnement des canaux pour les transmissions multidiffusions. Dans cette application, le premier groupe correspond à un ensemble de client et le deuxième groupe correspond à un ensemble de services demandé par ces clients. L'objectif de l'optimisation est de déterminer le nombre de sessions multicast permettant le partitionnement de l'ensemble des demandes. Ainsi, chaque service considéré doit appartenir à une session multicast tandis que chaque client peut être dans plusieurs sessions.

1.2.1 Definition le K-CmBCP

Nous considérerons un graphe biparti $G = (S, T, E)$ où S représente l'ensemble des services $i \in I$, T représente l'ensemble des clients $j \in J$, E représente l'ensemble des arêtes (i, j) du graphe biparti, $\bar{E}$ représente l'ensemble des arrêtes du graphe biparti complémentaire, c'est-à-dire que $\bar{E} = \{S \times T\} \backslash E$ et enfin, $K = \{1, ..., k\}$ qui représente l'ensemble des indices des K clusters prédéfinis dans le problème. Le graphe considéré étant non orienté, les clusters doivent induire une partition des services. En d'autres termes, chaque fois qu'un cluster est identifié, nous considérons toutes les arêtes reliant les services[1] appartenant à ce cluster, avec les clients correspondants. La figure 1.1 illustre le problème avec 2 clusters: 2-CmBCP.

[1]Dans notre cas, les sommets du côté gauche du graphe biparti, sont appelés les *Services* (S) et les sommets du côté droit du graphe biparti, sont appelés les *Clients* (T)

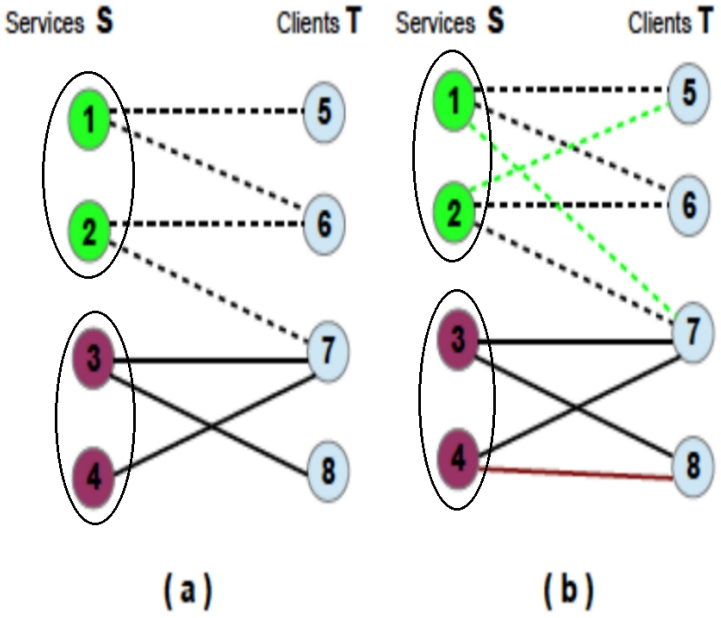

Figure 1.1: Exemple illustre le problème avec 2 clusters

La figure 1.1 est composée de deux parties: la partie (a), montre un graphe G bipartite avec les services $(S) = \{1, ..., 4\}$ et clients $(T) = \{5, ..., 8\}$ et deux possibilités de cluster $S_1 = \{1, 2\}$ et $S_2 = \{3, 4\}$. Les *arêtes* en pointillées appartiennent au premier *cluster* induit par S_1 et les *arêtes* en gras quant à elles, appartiennent au second *cluster* induit par S_2 et enfin, la partie (b), représente le graphe complémentaire G' permettant de former 2-bicliques.

La pénalité de ces 2-clusters est égale à $trois$, ceci est modélisé par deux arêtes manquantes dans le premier *cluster* et une arête manquante dans le deuxième cluster. Ainsi, les arêtes manquantes pour les deux clusters sont respectivement, les arêtes $(1, 7)$, $(2, 5)$, et $(4, 8)$.

L'existence de biclique contraint à ce que toutes ses arêtes soient présentes dans le graphe. Ainsi, les arêtes manquantes dans une solution réalisable génèrent des pénalités. Dans ce contexte l'objectif de la résolution est de trouver une façon de partitionner les services en minimisant les pénalités.

1.2.2 Modélisation du problème K-clusters dans un graphe biparti

Le modèle mathématique le plus utilisé du problème de K-clusters dans un graphe biparti peut être présenté comme suit:

$$x_{ik} = \begin{cases} 1 & \text{Si le service } i \text{ est couvert par le cluster } k \\ 0 & \text{Sinon} \end{cases} \tag{1.1}$$

$$y_{jk} = \begin{cases} 1 & \text{Si le client } j \text{ est couvert par le cluster } k \\ 0 & \text{Sinon} \end{cases} \tag{1.2}$$

soit z_{ijk} une variable binaire égale à 1, lorsque $x_{ik} = 1$ et $y_{jk} = 1$.

$$z_{ijk} = \begin{cases} 1 & \text{Si les noeuds } i \text{ et } j \text{ sont tous les deux affectés au cluster } k \\ 0 & \text{Sinon} \end{cases} \tag{1.3}$$

En utilisant les techniques de linéarisation sur le modèle mathématique le plus utilisé dans la littérature (vois [66,68,80,81]) on obtient la formulation suivante:

$$(ILPM): \quad \min \sum_{k \in K} \sum_{(i,j) \in \bar{E}} z_{ijk} \tag{1.4}$$

sous contraintes:

$$x_{ik} + y_{jk} \leq 1 + z_{ijk} \qquad \forall (i,j) \in \bar{E},\ \forall \text{ k} \in \text{K} \tag{1.5}$$

$$\sum_{k \in K} x_{ik} = 1 \qquad \forall \text{ i} \in S \tag{1.6}$$

$$x_{ik} \leq y_{jk}, \qquad \forall \text{ (i, j)} \in E,\ \forall \text{ k} \in K \tag{1.7}$$

$$x_{ik}.y_{jk} \in \{0,1\}, \qquad \forall \text{ i} \in S,\ \forall \text{ j} \in T,\ \forall \text{ k} \in K \tag{1.8}$$

$$z_{ijk} \in \{0,1\}, \qquad \forall \text{ } (i,j) \in \bar{E},\ \forall \text{ k} \in K \tag{1.9}$$

Où:

L'équation 1.4 représente la fonction-objectif qui minimise la pénalité totale de l'ensemble des K clusters. La contrainte 1.5 est la contrainte qui lie la variable z_{ijk} aux variables x_{ik} et y_{jk} en mettant z_{ijk} à 1 lorsque les deux autres le sont aussi. La contrainte 1.6 est la contrainte permettant d'attribuer chaque service $i \in S$ à un et un seul cluster. La contrainte 1.7 force chaque client $j \in T$ affecte au cluster k d'être dans le même cluster qu'un service $i \in S$ lorsque j est adjacent à i dans E. Et enfin, les contraintes 1.8 et 1.9 sont des variables binaires du problème.

1.2.3 Résolution hybride

Dans cette section, nous proposons de résoudre approximativement le K-CmBCP en utilisant des méthodes hydrides qui combinent les méthodes heuristiques et les méthodes de recherche par voisinage. Nous proposons de résoudre le K-CmBCP par application d'une hybridation de trois méthodes et qui s'appuie sur le principe de la coopération entre une recherche gloutonne, une intensification et une diversification. Cette approche utilise trois phases complémentaires :

- Phase 1 : La première phase construit une solution réalisable de départ.
- Phase 2 : La deuxième phase tente d'améliorer la qualité de la solution de départ par une intensification de la recherche dans une zone de l'espace de recherche.
- Phase 3 : La troisième et dernière phase introduit une diversification dans le but d'atteindre certains sous-espaces d'une manière efficace : dans cette phase des dégradations et reconstructions sont utilisées pour améliorer la qualité des solutions. Le processus utilisé dans cette phase est basé sur le principe de la recherche à large voisinage.

1.2.4 Phase de construction d'une solution de départ

Il existe plusieurs façons de construire une solution de départ, mais, nous avons proposé la procédure gloutonne basée sur le principe de l'algorithme séquentiel pour partitionner l'ensemble des services du graphique biparti et pour distribuer des services à leurs clusters par méthode séquentielle. La méthode séquentielle peut être décrite dans l'algorithme 1 qui résume la construction d'une solution de départ.

Algorithm 1 Construction d'une solution de départ

Entrées: Une instance de $k - CmBC$.
Sorties: Une solution de départ.

```
1:  s ← 1;
2:  while (chaque service s ≤ N) do
3:    k ← 1;
4:    for (chaque cluster k ≤ K) do
5:      Affecter le service s au cluster k
6:      Incrémenter s
7:      if (s > N) then
8:        Sortie
9:      end if
10:   end for
11: end while
```

1.2.5 phase d'intensification

Généralement l'amélioration de la qualité des solutions est réalisée par application de certaines procédures standards comme la recherche locale qui s'appuie sur des permutations des sommets et dont le but est d'effectuer une recherche sur des séries de voisinages. Dans notre cas, la phase d'intensification introduite est basée sur la stratégie de permutation de 2 éléments à la fois. Cette stratégie est notée par 2-opt. La procédure 2 présente la procédure d'améliorer de la solution de départ.

Algorithm 2 Recherche par voisinage pour améliorer une solution de départ.
Entrées: Une solution de départ.
Sorties: Une solution réalisable.

1: **while** (Le critère d'arrêt n'est pas effectué) **do**
2: `Procédure` – échang deux cluster;
3: Tri décroissant de tous les clusters K;
4: Tri décroissant de tous les services de chaque cluster;
5: $k_1 \leftarrow 1$, où $k_1 \in \{1, ..., K\}$;
6: **for** (chaque services $i = \{1...S1\}$), où $S1$ *désigne le nombre des services du cluster* k_1 ***do***
7: ***for*** *(chaque cluster* $q = \{K...k_2\}$*), où* $q \neq k_1$ ***do***
8: ***for*** *(chaque services* $j = \{S2...1\}$*),où* $S2$ *désigne le nombre des services du cluster* q ***do***
9: *échange du service* i *avec le service* j*;*
10: *Trouvez la nouvelle solution (*Sol'*);*
11: ***if*** $(Val(Sol') \leq Val(Sol))$ ***then***
12: $Sol \leftarrow Sol'$*;*
13: *Retri décroissant les services des clusters* k_1 *et* q*;*
14: ***else***
15: *Rechercher dans les voisinage du service* j*;*
16: ***end if***
17: ***end for***
18: ***end for***
19: ***end for***
20: `fin Procédure`
21: **end while**

Comme on peut le voir, le processus de classification et sélection est effectué de tous les clusters et leurs services avant le mouvement de les services entre deux clusters. Les classifications et le processus de sélection sont en fonction du nombre de bords manquants ajoutés pour compléter la solution. La stratégie de permutation est utilisée entre certains services des différents clusters que forment l'ensemble S. Un sous-ensemble de services voisins, qui peuvent être déplacés individuellement d'un cluster vers d'autres clusters. A partir de la meilleure

solution obtenue dans le voisinage, ce processus peut être réitéré. Ce dernier s'arrête après avoir exploré un certain nombre de voisinages ou après un nombre d'itérations paramétrable.

1.2.6 Phase de diversification des solutions

Dans cette section, nous montrons comment appliquer le principe de diversification pour rechercher une série de nouvelles solutions. La phase d'intensification peut améliorer les solutions si elle explore une série de voisinages très proches. Cependant, la reconfiguration de l'espace de recherche peut permettre l'exploration de nouveaux voisinages et donc l'émergence de meilleures solutions. La variante de la recherche dans un large voisinage introduite ici, est basée sur la stratégie de dégradation et la stratégie de reconstruction (Comme utilisée dans Hifi et Michrafy [53]). Après avoir appliqué les deux stratégies, la phase d'intensification est appelée de nouveau, afin d'améliorer la qualité de la solution courante.

Algorithm 3 dégradation et reconstruction stratégie

Entrées: Une solution réalisable Sol'. Max: est numéro d'itération maximum.
Sorties: Une solution approchée Sol_{global}

1: $Sol_{best} \leftarrow Sol'$;
2: **while** (Le critére d'arrêt n'est pas effectué) **do**
3: $Sol_{global} \leftarrow Sol_{best}$;
4: **while** $(LocalIteration < Max)$ **do**
5: $Sol_{local} \leftarrow Sol_{best}$;
6: *Détruire (α%)* de Sol_{local}, appliquer la stratégie de dégradation sur la solution courante Sol_{local};
7: *réparation* (Sol_{local}), appeler la stratégie de reconstruction pour compléter la solution partielle et Réaliser une nouvelle solution $(Sol_{current})$;
8: **if** $(Sol_{current} < Sol_{best})$ **then**
9: $Sol_{best} \leftarrow Sol_{current}$;
10: **end if**
11: **end while**
12: **end while**

1.3 le problème du sac à dos quadratique

Les problèmes du type sac à dos (noté: *Knapsack Problems*) ont été étudiés depuis plus d'un siècle (voir Matheows [29]), et plus intensément après les travaux de Dantzig (voir Dantzig [15]). Cela est dû au fait que non seulement ils possèdent de nombreuses applications pratiques importantes tant dans l'industrie que dans la gestion financière, mais aussi pour des raisons théoriques. En effet, ces problématiques se produisent fréquemment en tant que sous-problème dans les procédures de résolution de problèmes plus complexes (voir Hifi *et al.* [50]; Pisinger [16]), ce qui rend leurs modèles théoriques très importants pour la communautés scientifique (voir Pisinger *et al.* [17]).

Les problèmes de sac à dos quadratiques (noté QKP) contiennent une famille de problèmes de sac à dos classique avec une fonction objectif quadratique. Les problèmes de sac à dos quadratique est bien étudié que l'optimisation combinatoire et ont des nombreuses applications en théorie et en pratique. Une variété de procédures de limitation, heuristiques et algorithmes exacts sont disponibles pour cela. Bien que QKP n'ait pas été étudié aussi intensément, mais, des nombreux articles traitant du problème ont été présentés au cours des dernières années. Première introduction par Gallo et al. en 1980 [28] ont inventé le QKP et ont présenté une famille de limites supérieures en fonction des plans supérieurs, qui sont des fonctions linéaires des variables binaires que leur valeur est inférieure à la fonction objective QKP sur l'ensemble des solutions possibles.

1.3.1 Définition

Dans toutes les variantes du problème de sac à dos quadratique, pour un ensemble d'éléments donnés, les bénéfices ne sont pas seulement attribués à des objets individuels mais aussi à des paires d'entre eux. Le profit par paire est ajouté à la valeur d'objectif quadratique uniquement lorsque les deux éléments correspondants sont tous deux inclus dans le même sac à dos.

Soit un sac à dos quadratique avec une capacité fixe c et un ensemble de candidats objets (ou éléments), un ensemble $I = 1, 2, ..., n$, où chaque élément $i \in I$ est caractérisé par un positif poids w_i et un profit p_{ij}. La matrice entière positive $n \times n$, $P = (P_{ij})$ est supposée donnée, où $p_{ij} + p_{ji}$ est un profit obtenu en sélectionnant deux éléments différents i et j. Si élément i est sélectionné, il génère un profit p_{ij}. Le profit pair P_{ji} est ajouté à la valeur d'objectif quadratique uniquement lorsque l'objet j est inclus dans le même sac à dos et pour chaque paire d'objets i et j $(1 \leq i \neq j \leq n)$. L'objectif du problème est de sélectionner un sous-ensemble d'éléments de sorte que la somme des profits des éléments choisis soit maximum sans que la somme des poids ne dépasse la capacité c. La Figure 1.2 donne un exemple illustrant le problème du sac à dos.

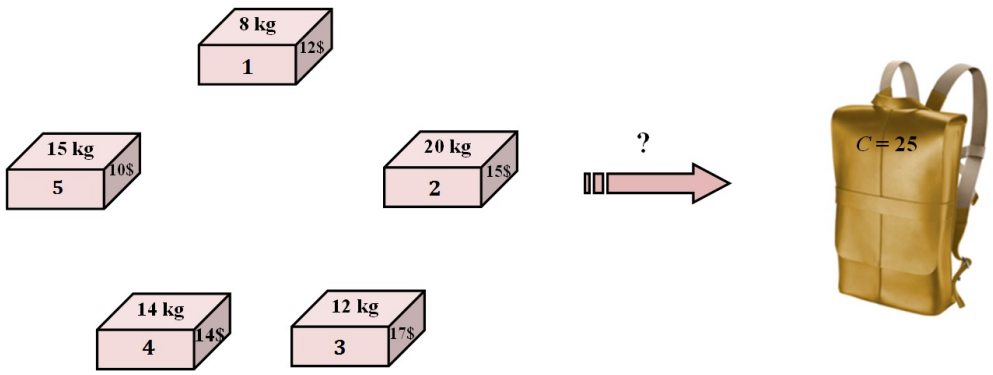

Figure 1.2: Example : illustre le problème du sac à dos

1.3.2 Modélisation du problème du sac à dos quadratique à variables binaires

Le problème du sac à dos à variables binaires est souvent représenté par le modèle simple suivant :

$$\max : \quad f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij} x_i x_j \tag{1.10}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_j \leq c \tag{1.11}$$

$$x \in \{0,1\} \qquad \forall i \in I = \{1, ..., n\} \tag{1.12}$$

où x_i représente la variable de décision telle que :

$$x_i = \begin{cases} 1 & \text{si l'objet } i \text{ est placé dans le sac} \\ 0 & \text{sinon} \end{cases}$$

Ce modèle est représenté par une fonction objectif (Eq. 1.10), une contrainte dite de *knapsack* ou de capacité (Eq. 1.11) et les contraintes d'intégralité sur les variables de décision (Eq. 1.12). L'objectif du problème est de sélectionner un sous-ensemble d'éléments, permettant de maximiser la somme totale des profits tout en satisfaisant la contrainte de capacité. Notons qu'afin d'éviter les cas triviaux, on suppose que :

- Toutes les valuers $c, p_i, w_i, \forall i = 1, \ldots, n$, sont des entiers positifs.
- $\sum_{i \in n} w_i > c$, permettant d'éviter les solutions triviales.

1.3.3 Recherche réactive pour résoudre le problème du sac à dos quadratique

Dans cette section, nous présentons le premier algorithme réactif pour résoudre le problème du sac à dos quadratique. Cet algorithme comporte deux méthodes: une première méthode gloutonne qui ignore la contrainte quadratique et une deuxième méthode qui comporte deux étapes complémentaires: une première étape de destruction et une deuxième étape de reconstruction. Cet algorithme peut être vu comme une méthode de recherche aléatoire par voisinage.

1.3.4 Méthodes glouton pour construction d'une solution de départ

Parmi les méthodes heuristiques, nous citons une méthode basée sur le principe glouton (voir Dasgupta *et al.* [78]). Les méthodes gloutonnes des méthodes

heuristiques qui construisent une solution en se basant sur une amélioration courante. C'est-à-dire que, l'algorithme choisit toujours un optimum local dans l'espoir de trouver un optimum global dans la suite des développements.

Dans ce qui suit, nous présentons, une méthode gloutonne basée sur les principes proposés par Dantzig (voir Dantzig [15]) pour résoudre le 0-1 KP.

Premièrement, tous les éléments sont triés par ordre décroissant en fonction de leur rapport du profit sur le poids (p_i/w_i), tel que:

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_n}{w_n}$$

Ainsi, à chaque étape, sélectionner de façon gloutonne un élément selon l'ordre défini précédemment. Si l'élément est recevable, cela veut dire si son poids ne dépasse pas la capacité du sac après fixation des autres éléments, alors, il est placé dans le sac à dos. Sinon, nous sélectionnons l'élément suivant qui peut être placé, et ainsi de suite, jusqu'au remplissage du sac à dos. (voir Algorithme 4).

Algorithm 4 Une méthode gloutonne pour QKP

Entrées: Une instance de QKP.
Sorties: Une solution de départ S.

1: Soit c le capacitéde sac à dos;
2: Tri décroissant $(e_i = \frac{p_i}{w_i})$;
3: $Mettre\ Z \leftarrow 0$ (profit);
4: **for** (chaque élément $i \leq N$) **do**
5: **for** (chaque élément $j \neq i$) **do**
6: **if** $(w_i \leq c)$ **then**
7: $x_i = 1$;
8: $c = c - w_i$;
9: $Z = Z + P_{ij} + P_{ji}$;
10: **else**
11: $x_i = 0$;
12: **end if**
13: **end for**
14: **end for**

Il convient de noter que, cette méthode n'est pas la seule méthode gloutonne pour le problème de sac à dos. Ainsi, il existe plusieurs autres versions et améliorations par rapport à cette simple méthode (voir Kellerer *et al.* [33]).

1.3.5 Recherche par voisinage pour améliorer la solution de départ

Dans cette section nous pouvons améliorer la solution de départ en utilisant une recherche par voisinage qui combine deux stratégies: *degradant* et $re-construire$ stratégies (comme utilisé dans Hifi et Michrafy [52]), mais, nous avons amélioré

ce processus en réajustant la solution globale et en intégrant une liste d'éléments à ne pas inclure dans la sélection. Dans ce qui suit, nous montrerons comment combiner les deux stratégies pour la recherche d'une série de nouvelles solutions. Soit S la solution de départ obtenue à partir de l'algorithme 4. Soit α une constante, de sorte que $\alpha \in [0, ..., 100]$, indiquant le pourcentage des éléments emballés de l'ensemble N, soit certains éléments sont chargés dans le sac à dos selon la solution actuelle S. Ensuite, la procédure de diversification peut être considérée comme une autre approche qui est réalisée en utilisant deux procédures qui sont:

- La stratégie **degradant** est utilisée pour fixer $\alpha\%$ articles de la solution actuelle pour extraire une solution partielle $S^{'}$.
- La stratégie **re − construction** est utilisée pour compléter la solution partielle obtenue après la destruction et ré-optimiser la solution complémentaire $S^{''}$ par application le solveur Cplex sur le sous-ensemble $S^{''}$ et pour améliorer la solution actuelle. Cette procédure est basée sur la destruction d'un certain nombre d'éléments de la solution actuelle et remplacée par les éléments de ses voisins dont le poids total ne dépasse pas une capacité de knapsack donnée (voir. Algorithme 5).

Algorithm 5 L'algorithm de recherche par voisinage.

Entrées: Une solution de départ S 0-1 QKP.
Sorties: Une meilleure solution S_{global}.

1: $S_{global} \leftarrow$ S.
2: **while** (Le critère d'arrêt n'est pas effectué) **do**
3: $S_{local} \leftarrow S_{global}$
4: **while** $(I < Max_{Iteration})$ **do**
5: Appeler la procédure de dégradation afin de fournir une solution partielle $S^{'}$
6: Appeler ré-optimiser la solution complémentaire $S^{''}$ pour réaliser une nouvelle solution S_{local}
7: **if** $(Val(S_{local}) > Val(S_{global}))$ **then**
8: $S_{global} \leftarrow S_{local}$
9: **end if**
10: **end while**
11: **end while**

Du travail présenté dans cette thèse, il faut attirer l'attention sur l'objectif principal considéré lors de la mise en route de cette recherche.

En effet, l'objectif principal était de développer des approches basées sur des techniques de recherche par voisinage conçues pour optimiser les cas problèmes de grande taille. Le résultat général des analyses expérimentales que nous avons

mené sur les exemples de référence de la littérature, montre que les techniques de recherche par voisinage réussissent à développer des solutions de bonne qualité.

Afin de développer notre algorithme, nous proposons une autre direction de recherche pour résoudre les problèmes de knapsack quadratique (en particulier pour les instances de grande taille), à travers la recherche de réduction des variables 1.3.3. Nous présenterons ce travail dans la section suivante.

1.3.6 La réduction de variables

Dans cette section, nous proposons une réduction de variables pour résoudre les problèmes de grande taille. A partir du travail présenté dans la section 1.3.3 et selon l'analyse expérimentale sur toutes les instances de référence de la littérature, il est nécessaire d'introduire un certain développement sur notre algorithme pour fournir une méthode de résolution efficace fournissant des solutions de haute qualité avec un temps d'exécution rapide.

Les résultats expérimentaux montrent l'efficacité de la recherche réactive proposée pour fournir des solutions de bonne qualité. Les résultats expérimentaux montrent que l'algorithme proposé est capable de résoudre les très grandes instances qui ne peuvent être résolues par le solveur Cplex (voir Algorithme 6).

La figure suivante montre le principe algorithmique de la réduction des variables que nous avons appliqué.

1.4 Conclusion

Nous avons proposé dans le cadre de cette thèse plusieurs méthodes de résolution hybrides et approchées pour résoudre deux problèmes combinatoires. La première partie de la thèse est consacrée au problème de k-clusters dans un graphe bipartite. La deuxième partie traite le problème de sac à dos quadratique. Les deux problèmes sont difficiles à résoudre d'un point de vue informatique et sont très utiles dans de nombreux domaines scientifiques et applicatifs. Les résultats expérimentaux que nous avons menés sur des instances connues de la littérature montrent l'efficacité des méthodes proposées dans l'optimisation des problèmes de grande taille. Plusieurs pistes d'amélioration et de perfectionnement restent possibles dans la mécanique interne des méthodes proposées. Voici quelques pistes qui méritent d'être explorer à court terme. Pour le problème de k-clusters dans un graphe bipartite : - Etendre la sélection à plusieurs éléments en cours de résolution dans la deuxième phase de l'algorithme hybride - Rendre la sélection du voisinage plus intelligente - Explorer le développement parallèle de plusieurs partitionnements. Pour le problème de sac à dos quadratique : - Améliorer le principe de permutations et l'étendre à plusieurs éléments à la fois. - Améliorer la fixation (réduction du problème) et concevoir une méthode complémentaire avec la possibilité de faire des retours arrières plus maitrisés lors de la résolution.

Algorithm 6 Réduction des variables de la solution de départ S.

Entrées: Une solution de départ S 0-1.

Sorties: Une meilleure solution S_{global}.

1: $S_{global} \leftarrow$ S.
2: **while** (Le critère d'arrêt n'est pas effectué) **do**
3: $S_{local} \leftarrow S_{global}$.
4: **while** ($I < Max_{Iteration}$) **do**
5: $\alpha \leftarrow [0, ..., 100]$.
6: $\beta \leftarrow [0, ..., 100]$.
7: Soit X_j comme ensemble des élément emballés associé à la solution courante S_{local}.
8: Sélectionner $\alpha\% \in X_j$ appartenant à la solution actuelle S_{local}, où $X_j = 1$.
9: Sélectionner $\beta\% \in X_j$ appartenant à la solution actuelle S_{local}, où $X_j = 0$.
10: Fixer $\alpha\%$ des variables X_j évalués à 1.
11: Fixer $\beta\%$ des variables X_j évalués à 0.
12: **if** ($\alpha\% > 0$) et ($\beta\% > 0$) **then**
13: Extraire la solution partielle $S^{'}$
14: Trouve la solution complémentaire $S^{''}$
15: Appeler ré-optimiser la solution complémentaire $S^{''}$ pour déterminer une nouvelle solution S_{local}.
16: **if** ($Val(S_{local}) > Val(S_{global})$) **then**
17: $S_{global} \leftarrow S_{local}$.
18: **end if**
19: **end if**
20: **end while**
21: **end while**

Chapter 2

Chapter 2

General Introduction

There are numerous practical situations that can be formulated as combinatorial optimization problems. Among these ones, we interested with some problems such as the K-clustering minimum bi-clique completion problem (K-CmBCP) and the quadratic knapsack problem(QKP), that are very significant in real world and that have been applied to several real applications. On the other hand, these problems are highly combinatorial and difficult to solve from computational perspective.

In order to solve such problems two directions of researches can be followed: (i) Find the optimum solutions for a given problem using exact methods. (ii) searching near optimal solutions using approximate methods. The basic principle of an exact algorithm is in general, the enumeration of the set of solutions in the search space implicitly. Often, this type of algorithms solves the small problems only. Otherwise, the computing time increases exponentially with the size of the problem. However, for extensive problems, an exact method may be expended exponential computing time. This often induces to large computing time for the practical situation. Thus, the evolution of heuristic methods has accounted for more consideration in the last decades. So that, there are other solution methods that combine several resolution methods such as (heuristic, meta heuristics and exact), these methods, known as hybrid methods, that are approximate methods more powerful than the other methods using for solving combinatorial optimization problems.

Conversely, the hybrid approximate methods are designed to produce solutions of good quality with a reasonable resolution time, but not necessarily optimal. Thus, the approximate methods local heuristics, local searches that give a sequence of solutions up to the local optimum.

So that, our work focuses on using the hybrid methods for approximating a particular problems such as the K-clustering minimum bi-clique completion problem and the quadratic knapsack problem. The K-clustering minimum bi-clique completion problem (K-CmBCP) has many applications, especially in the field of telecommunication, since telecommunication network cannot manage many multicast sessions at the same time, it is hence necessary to group the sessions into a limited number of clusters. Nowadays, operational research on such problems is very important because it allows the design of critical information systems in a decision making. Indeed, these systems are used to model and treat the company's information flow in order to help decision making knowing that the ultimate goal is to satisfy customers, within the constraints and, at lower cost.

On the other hand, the quadratic knapsack problem can be used to accommodate a wide range of practical applications in numerous domains especially in transport logistics.
This thesis presents two hybrid methods based upon neighborhood search techniques. These methods are designed for optimizing the large size instances of a combinatorial problems. The first hybrid method is to solve the (K-CmBCP), although such technique produce approximate solution, it allows us to present fast algorithm that yield interesting solutions within a short average running time.
Furthermore, we introduced the second hybrid method to solve the quadratic knapsack problem that is mainly based on two complementary phases. The method combines the principle of a neighborhood search and a strategy of variables reduction search. Although such technique produces approximate solutions, it is capable of solving the small, large and very large size instances of the QKP that cannot be solved by Cplex solver and by other methods.
Consequently, our work may be interested in the progress of optimization techniques that can be applied to handle large size instances of different combinatorial optimization problems.

Thesis organization

The thesis contains two main parts. The first part of this thesis consists of two chapters devoted to study of the K-clusters in a bipartite graph. The second part of this thesis consists of three chapters devoted to study of the quadratic knapsack problem.

More specifically, this thesis is organized as follows. The first part(chapter 3) presents the state of the art for K-clusters in a bipartite graph problems, which is after finding the number of K-clusters that minimizes the number of added edges to get K bi-cliques. First, it begins by recalling some graph notions theory in order to establish the notation that will be used throughout of this part of the thesis. Second, a brief overview of the K-clusters problem in a bipartite graph is presented. Third, details of the resolution methods used in the literature to solve this problem, some examples of K-CmBCP and the main mathematical models. And finally, we illustrate our proposed method to solve this problem. Chapter 4, presents a hybrid method that combined heuristic and the neighborhood search methods to approximately solving the problem of K clusters in the bipartite graph.

The second part of this thesis contains three chapters: (chapter 5, chapter 6 and chapter 7). In chapter 5, we discuss some of methods used in the literature to solve the combinatorial optimization problems. These methods can be classified into two categories: methods which guarantee the optimality of solution and the approximated methods (heuristic and meta heuristic) that give approximate solutions.

In chapter 6, we present our first reactive algorithm to solve the quadratic knapsack problem. Herein we use a combination of a greedy algorithm and a neighborhood search techniques in order to produce quick solutions with high quality. This algorithm is composed of two phases : the first phase serves to provide the starting solution using the greedy algorithm based on the principles of efficiency items for filling the knapsack where all items are sorted by decreasing order as a function of the profit to weight ratio ($e_i = p_i/w_i$). While the second phase serves to improve the quality of starting solution using two complementary strategies: a first strategy for destruction and the second strategy for re-optimization. The destroy strategy performs randomly in order to produce a new neighborhood solution. This strategy is very important, where it controls the efficiency of algorithm and increases the search space. Moreover, it helps to escape from a series of local optimum. Actually, the solution produced by the destroy strategy is processed as a reduced problem, that will be re-optimized. Indeed, re-optimizing strategy consists of two procedures, the first procedure serves to provide a feasible solution while the second procedure serves to improve the quality of such solution.
Chapter 7 presents the second hybrid reactive algorithm to solve the quadratic knapsack problem. In fact, the goal is to propose an efficient algorithm that is capable of guiding the search process towards new research spaces with high quality solutions. This method combines the principle of the large neighborhood search and the variables reduction strategy. More precisely, the reduction strategy is used to yield a sub-solution space, while, LNS is then applied in order to explor and improve the current local optimum in its neighborhood.
However, the solution space of the quadratic knapsack problem might be too large, to be efficiently searched by an algorithm, the size of the quadratic knapsack problem may be greatly reduced using some reduction techniques such that, one may obtain a feasible solution by truncating some variables from the search space [73]. In this chapter, to further reduce the search space to explore, we fix some variables to a manageable size. Although such technique produces approximate solutions, it is capable of solving the small, large and very large size instances of the QKP that cannot be solved by Cplex solver and by other methods. The performance of this hybrid algorithm is tested and evaluated in this chapter.

Finally, chapter 8 summarizes the general conclusion obtained and indicates some viewpoints for the future works in this area.

Part I

The problem of K-clusters in a bipartite graph

Chapter 3

Chapter 3

Literature Review

Contents

This chapter presents a state of the art of the problem K-clusters in a bipartite graph, which consists of finding K- clusters that minimizing the number of edges to be add to get K bi-cliques. This problem is known as K-clustering minimum biclique completion problem (noted: K-CmBCP) and it has been shown to be NP-hard (see Gualandi et al. [80]).

In this chapter, we begin by recalling some notions of the graph theory that will allow to establish the notation used throughout this part of the thesis.

Then, the section 3.2 will give an overview of the problem K-clusters in a bipartite graph. Section 3.3 will present the resolution methods used in the literature to solve this problem and some examples of K-CmBCP. Section 3.4 will detail the main mathematical models. And finally, Section 3.5 will summarize this chapter and introduce the resolution method we proposed for this problem.

3.1 Graph notions

An undirected graph G (generally called simply *graph*) is a pair (V, E) where V is a finite set of vertices, and $E \subseteq V \times V$ is a finite set of edges (pairs of vertices $(i, j) \in V \times V$). An edge is a pair of vertices that are called the *endpoints* of the edge.

A *graph* can be directed or not:

- In a directed graph, the pairs $(i, j) \in E$ are directed, ie (i, j) is an ordered pair, where i is the initial vertex, and j is the terminal vertex. In this case, a pair (i, j) is called an arc, and is represented graphically by ($i \longrightarrow j$).
- In an undirected graph, the pairs $(i, j) \in E$ are not directed ie (i, j) is equivalent to (j, i). In this case, a pair (i, j) is named an edge, and is graphically represented by $(i \_ j)$.
  However, we shall refer to undirected graphs only, since our work is based on this notion (see [13]).

Terminology

- Two vertices are *adjacent* or *connected* if there is an edge between them. It is up to a vertex i is said to be adjacent (or neighbouring) to another vertex j, if there exists an edge between i and j. In undirected graph $G = (V, E)$, the *neighborhood* of a vertex $i \in V$, is denoted $N_G(i) = \{j : (i, j) \in E\}$.
- The *order* of a graph is the number of its vertices $|V|$.
- The degree $d(i)$ of a vertex i is the number of edges connecting this vertex to the others neighbour.
- $G' = (V', E')$ is called a sub-graph of undirected graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.
- Undirected graph $G = (V, E)$ is said to be *connected*, whatever the vertices i and j of V, there exists a chain from i to j. That is, if there exists a sequence of edges to reach j from i (or i from j).
- Undirected graph is said to be *complete*, if for all the pairs of distinct vertices i and j there exists an edge between i and j. We can also say that a graph is complete, if all its vertices are adjacent.
- Undirected graph $G = (V, E)$ can be equivalently described by its *adjacency matrix* $\mathbf{A}$ defined by:

$$\mathbf{A}\left[i, j\right] = \begin{cases} 1 & \text{If } (i, j) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{3.1}$$

- A *clique* in an undirected graph $G = (V, E)$ is a subset of vertices set of this graph whose induced sub-graph is complete, that is to say that two peaks of the clique are always adjacent. The size of a clique is the number of vertices it contains.
- A graph $G = (V, E)$ is called a *bipartite* graph if the set of vertices V can be partitioned into two disjoint non-empty subsets S and T such that, $V = S \cup T$ and $S \cap T = \emptyset$ and every edge in E connects a vertex in S and a vertex in T. The *bipartite* graph G is denoted by $G = (S, T, E)$ where, S

and T, in our case, are called *services* and *clients* respectively. It should be noted that in one bipartite graph, there is no edge in E connecting two vertices within S or two vertices within T.

- The *complete bipartite* graph $G = (S, T, E)$ is a *bipartite* graph that contains the maximum number of edges. This means that, $\forall\ i \in S$ and $\forall\ j \in T$, there is an edge between i and j. A *complete bipartite* graph is also called a *biclique*.

- In the problem of K-clusters in a bipartite graph, a *cluster* is a sub-set of services $S_1 \subset S$, such that $|S_1| \geq 1$ service.
  If we consider a bipartite graph $G = (S \cup T, E)$, where S and T are two sets of vertices of the bipartite graph and E is the set of edges between S and T. The cluster $\{S_1, T_1\}$ with $S_1 \subset S$ and $T_1 \subset T$ is a *biclique* if the sub-graph induced from G by $S_1 \cup T_1$ is complete.

To transform K-clusters into K bicliques, we can add or remove edges. In the case of the problem of K-clusters, the objective is to find the minimum number of edges that must be added to form K bicliques [66].

3.2 Definition of the problem of K-clusters in a bipartite graph

The problem we are dealing with in this part of the thesis is the K-Clustering minimum Biclique Completion Problem denoted: K-CmBCP, defined on bipartite graphs. The goal of solving this problem is to group the nodes of the services into K-clusters, so that the number of edges that must be added to form K bicliques is minimum. Since the graph considered is undirected, the clusters must induce a partition of the nodes of services. In other words, whenever a *cluster* is identified, we consider all the possible edges connecting the services that belonging to this cluster, with clients corresponding. We note that a *biclique* requires that all its edges be presented in the graph, thus, the missing edges in a feasible solution are considered as penalties. The objective is to find the way to partition the nodes of services in order to have the lowest total cost or penalty. Figure 3.1 illustrates the problem with 2 clusters: 2-CmBCP.

Thus, Figure 3.1 is composed of two parts: part (a), shows a bipartite graph G with the services $(S) = \{1, ..., 4\}$ and clients $(T) = \{5, ..., 8\}$, and two possibilities of cluster $S_1 = \{1, 2\}$ and $S_2 = \{3, 4\}$. The dashed edges belong to the first *cluster* induced by S_1 and the edges in bold, belong to the second *cluster* induced by S_2 and finally the part (b) represents the complementary graph G' allowing to form 2-bicliques.

The penalty of these 2-clusters is equal to *three*, this is given by *two* missing edges in the first *cluster* and *one* missing edge in the second *cluster*. Thus, the missing edges for two clusters, that are, respectively: $(1, 7)$, $(2, 5)$, and $(4, 8)$, [55].

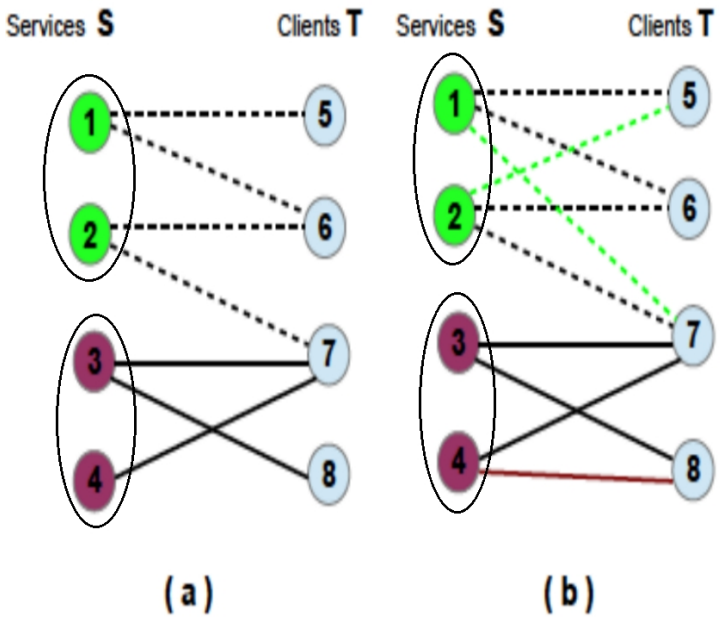

Figure 3.1: An example of K-CmBCP for 2-cluster in a bipartite graph G.

In the literature, there are several problems concerning the *bicliques* in a bipartite graphs. Some use a single biclique, this is the case of the problem consisting of finding the maximum biclique in number of edges (noted: MEB[1]) which has been proved NP-complete in Peeters [74].

However, other problems take into account several *bicliques* in the bipartite graph. For example, the problem of coverage by a minimum of *bicliques*, known as the "*minimum biclique cover*", which consists of finding a set of *bicliques* of G such that each edge $e \in E$ belongs to at less a *biclique*. The problem of coverage by a minimum of biclique, is also NP-complete (see Orlin [43]).

In the next section, we provide an overview on the resolution methods and applications for K-CmBCP.

3.3 The resolution methods and applications for K-CmBCP

In this section, first we will give the few solving methods we have found in the literature to deal with the problem of K-clustering in a bipartite graph. And then, we will give the fields of application for this problem. To our knowledge, the K-CmBCP is very little studied in the literature, only some papers concerning it are available in the literature to date. Among these papers we mention:

[1]MEB:= Maximum Edge Biclique problem

The paper of Faure *et al.* [68] in which the authors proposed a model of linear programming to solve small-sized instances with optimality and in the same paper, the authors also proposed a heuristic based on columns-generating approach, where certain instances of large sizes have been resolved.
The paper of Gualandi [80] which addressed the problem of K-CmBCP using a hybrid approach that combines constraint programming and semidefinite programming exact approach. It should be noted that although the method proposed in this paper is outside the scope of our study, it is one of the few works on this problem, which should be mentioned.
To our knowledge, the method introduced by Gualandi *et al.* [81], based on the principle of branch and price to accelerate the search process and to improve the quality of solutions obtained by Cplex solver that based on the mathematical model (ILP) to solve K-CmBCP.
Concerning the fields of application, the problem of K-clusters in a bipartite graph is the basis of several industrial applications that have brought the attention of researchers deals with telecommunications. Thus, one of the applications that made K-CmBCP known to researchers is the problem of aggregation of multicast sessions in telecommunication. In this case, a multicast session is defined as a subset of clients requiring the same information. Also, each client can require several multicast sessions. Since the telecommunication network cannot manage many multicast sessions at the same time, it became necessary to collect number of sessions in the limited clusters. The problem then consists in aggregating the sessions into clusters to limit the number of unnecessary information sent to clients.
The variant we deal with in this part of the thesis requires the clusters to define a partition on the set of sessions. This variant is inspired by problems like the *Multicast Partition Problem: MPP* (see Suh *et al.* [64]). Since this application was the first one that brought the K-CmBCP to attention, Figure 3.2 illustrates an example of the *MPP* in which we use the same representation as the K-CmBCP. Thus, on the left side of the bipartite graph, there are three sessions A, B and C, which send multicast information to clients 1, 2, 3 and 4 (on the right side of the bipartite graph). Each client requests a certain number of sessions which are mentioned on its right.

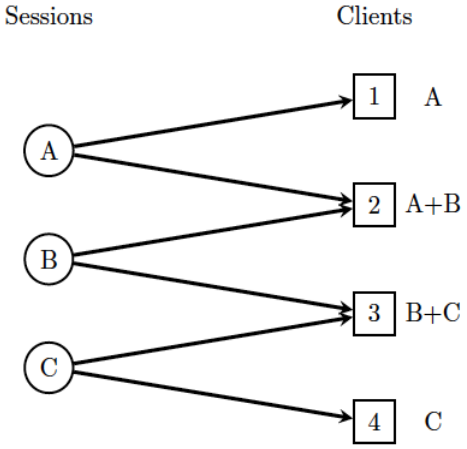

Figure 3.2: An example of the MPP problem .

Figure 3.3 shows the same example of the MPP, by grouping sessions A and B sets and which now become a single multicast source allowing to send all the information that sessions A and B sent separately. Thus, client 1 receives session B although it does not request it. The same can be said for client 3 with session A. It is noted that this union, allowed the gain of a link while respecting the requests of the customers.

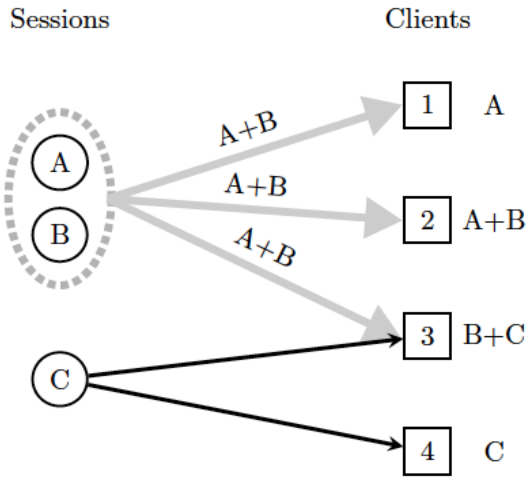

Figure 3.3: Two sessions are grouped in cluster for the previous MPP example.

Other than MPP, applications of K-CmBCP exist in domains such as: computational biology, the network security analysis or *data mining* where *clustering* or *grouping* is a very important method.

The following section gives the main mathematical formulations for the problem of K-clusters in a bipartite graph.

3.4 Mathematical programming formulations for (K-CmBCP)

This section is devoted to the description of different mathematical formulations of cluster problem in a bipartite graph as shown in Magni *et al.* [66]. The problem of K-clusters in a bipartite graph is a problem of optimization combinatorics with several types of mathematical formulations:

1. Assignment formulations, in which clusters are indicated explicitly and the vertices are assigned to these clusters.
2. Representative formulations, where clusters are indicated by a representative node that is noted "Head" and all other vertices are assigned to their vertex.
3. Column generation formulations, where configurations of clusters are considered and the number of variables is exponential.

In the following sections of this chapter, we will present five mathematical formulations for K-clusters problem in a bipartite graph. For all the mathematical formulations, we will consider: A bipartite graph $G = (S, T, E)$ where S represents the set of services i and T represents the set of clients j. E represents the set of edges $(i,\ j)$ of the bipartite graph, $\bar{E}$ represents the set of edges of the complementary bipartite graph, i.e $\bar{E} = \{S \times T\} \backslash E$ and finally $K = \{1, ..., k\}$ represents the set of indices of the predefined K-clusters in the problem.

In order to obtain clarity, we use the terminology of the Magni *et al.* [66] and Gualandi *et al.* [80].

3.4.1 First formulation

This first formulation is certainly the most obvious and the most used in the literature, Faure *et al.* [68] and Gualandi *et al.* [80, 81], and it is probably the most immediate to derive. It is quadratic, with both linear and non linear constraints. The binary variables are:

$$x_{ik} = \begin{cases} 1 & \text{if the service } i \text{ belong to cluster } k \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$y_{jk} = \begin{cases} 1 & \text{if the client } j \text{ belong to cluster } k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

When the service i and client j are in the cluster k, and if $x_{ik} = y_{jk} = 1$ and the edge $(i, j) \notin E$ (or $(i, j) \in \bar{E}$), then, the edge (i, j) is counted as a penalty.

The objective function is obtained by the sum of all the edges that are added, thus they can be expressed in the following equation:

$$\min \sum_{(i,j)\in\bar{E}} \sum_{k\in K} x_{ik}y_{jk} \tag{3.4}$$

with constraints:

$$\sum_{k\in K} x_{ik}y_{jk} = 1 \qquad \forall\ (i,j) \in E \tag{3.5}$$

$$\sum_{k\in K} x_{ik} = 1 \qquad \forall\ i \in S \tag{3.6}$$

$$\sum_{i\in S} x_{ik} \geq 1 \qquad \forall\ k \in K \tag{3.7}$$

$$x_{ik}, y_{jk} \in \{0,1\} \qquad \forall\ i \in S,\ j \in T,\ k \in K \tag{3.8}$$

The first constraint (3.5) expresses the covering of all the demands. For example, each edge (i,j), must ensure that service i and client j belong to one cluster at most. The second constraint (3.6) expresses the partition of services in clusters. This constraint requires that a service i belong to a single cluster at a time. The third constraint (3.7) ensures that there are no empty clusters, requires the use of all K-clusters and finally the fourth and last constraint (3.8) denotes the binary variables.

The above model of this type with non-linear constraints is very difficult to solve. However, by keeping the constraint (3.6) and by combining the constraints (3.5) and (3.7), the model can be linearised by introducing a third binary variable z_{ijk} that expresses the product of $x_{ik}y_{jk}$. In this case, the new binary variable will be:

$$z_{ijk} = \begin{cases} 1 & \text{if the service } i \text{ and the client } j \text{ are both in cluster } k \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

and the mathematical model can be rewritten under a quadratic formulation, with linear constraints . Thus, the obtained linear formulation can be stated as follows :

$$\min \sum_{(i,j)\in\bar{E}} \sum_{k\in K} z_{ijk} \tag{3.10}$$

with constraints:

$$y_{jk} \geq x_{ik} \qquad \forall\ (i,j) \in E,\ \mathrm{k} \in K \tag{3.11}$$

$$\sum_{k \in K} x_{ik} = 1 \qquad \forall \, \mathrm{i} \in S \tag{3.12}$$

$$z_{ijk} \geq x_{ik} + y_{jk} - 1 \qquad \forall \, \mathrm{i} \in S, \;\; j \in T, \;\; k \in K \tag{3.13}$$

$$x_{ik}, \;\; y_{jk}, \;\; z_{ijk} \in \{0, 1\} \qquad \forall \, \mathrm{i} \in S, \;\; j \in T, \;\; k \in K \tag{3.14}$$

In this last formulation, the constraint (3.11) makes it possible to cover all the applications. The constraint (3.12) assigns each service i of S to a single cluster at a time. The constraint (3.13) is used to present the linear equation, and finally the last constraint (3.14) denotes the binary variables of the problem.

3.4.2 Second formulation

Here, we present a different non-linear formulation that interprets the problem as a *min-max* model (see Meka [7], Magni *et al.* [66] and Gualandi *et al.* [80]). The x_i variables are the same as the previous model, but a variable z_{ij} introduced as the following:

$$z_{ij} = \begin{cases} 1 & \text{if the edge } (i,j) \text{ is added.} \\ 0 & \text{Otherwise} \end{cases} \tag{3.15}$$

Note that an edge $(i, j) \in \bar{E}$ is added or is counted as a penalty if and only if there is at least a service l, connected to the client j and the service l is in the same cluster k as service i.

On the other hand, if $N(j)$ indicates the neighbourhood of clients j, such that $N(j) = \{i \mid (i, j) \in E\}$, so, this constraint can be written as:

$$z_{ij} = \max_{l \in N(j)} \{x_{ik} \; x_{lk}\} \qquad \forall \, \mathrm{l} \neq i \tag{3.16}$$

This constraint can be rewritten as a set of constraints
$z_{ij} \geq x_{ik} \; x_{lk}$, which are easily made linear with: $z_{ij} \geq x_{ik} + x_{lk} - 1$.

So, the linear formulation for the problem K-clusters in a bipartite graph using the *Min-Max* model is:

$$\min \sum_{(i,j) \in \bar{E}} z_{ij} \tag{3.17}$$

with constraints:

$$\sum_{k \in K} x_{ik} = 1 \qquad \forall \, \mathrm{i} \in S \tag{3.18}$$

$$z_{ij} \geq x_{ik} + x_{lk} - 1 \qquad \forall \text{ (i, j)} \in \bar{E},\ \ (l,j) \in E,\ \ k \in K \tag{3.19}$$

$$x_{ik},\ z_{ij} \in \{0,1\} \qquad \forall \text{ i} \in S,\ \ j \in T,\ \ k \in K \tag{3.20}$$

Constraint (3.18) expresses the partition of services, while constraint (3.19) is the linearization of the *min-max* (3.16) constraint defined above and the constraint (3.20) gives the binary variables of the problem.

3.4.3 Third formulation

The third model uses concept of *representative* as the node identifying a cluster. Thus, service node is selected as representative of a cluster and all other services belonging to the same cluster must express their assignment to the representative. The construction is done in any order. So, in this model, the first service of each cluster becomes its representative.

This model is useful to eliminate symmetries. The binary variables for this model are:

$$x_{il} = \begin{cases} 1 & \text{if the service } i \text{ is representative of the service } l \\ 0 & \text{Otherwise} \end{cases} \tag{3.21}$$

$$z_{ij} = \begin{cases} 1 & \text{if the edge } (i,j) \text{ is added} \\ 0 & \text{Otherwise} \end{cases} \tag{3.22}$$

The objective function is straightforward, thanks to the variable z. But, we need a constraint to ensure that K clusters are identified. And other one to assign each service to *a single representative*, thus to a single cluster. Another constraint is required to correctly link the variables x to each other (that is, if a service l is assigned to a representative i, then i must be a *representative* and cannot be a simple node). The last constraint for setting the binary variable z_{ij}; it must be set to 1 if the edge (i,j) is to be added to complete a cluster (i.e. to obtain a *biclique*).

$$\min \sum_{(i,j) \in \bar{E}} z_{ij} \tag{3.23}$$

with constraints:

$$\sum_{i \in S} x_{ii} \ = \text{K} \tag{3.24}$$

$$\sum_{i \in S,\ i \leq l} x_{il} \ = 1 \qquad \forall\ l \in S \tag{3.25}$$

$$x_{il} \leq x_{ii} \qquad \forall\ i \in S,\ l \in S \tag{3.26}$$

$$z_{ij} \geq x_{hi} + x_{hl} - 1 \qquad \forall\ (i,j) \in \bar{E},\ h \in S,\ (l,j) \in E \tag{3.27}$$

$$x_{il},\ z_{ij} \in \{0,1\} \qquad \forall\ i \in S,\ l \in S,\ j \in T \tag{3.28}$$

Constraint (3.24) ensures that we actually group the services in K-clusters. Constraint (3.25) ensures that each service has one and only one *representative*, which can be translated as each service being in exactly one cluster. Constraint (3.26) forces a service that is representative of another one to be representative of itself too. The constraint (3.27) forces creating an biclique, when the right condition is met, the last constraint(3.28) gives the binary variables of the problem, (see Magni *et al.* [66] and Gualandi *et al.* [80, 81]).

3.4.4 Fourth formulation

In this model the clusters are considered as combinatorial objects.

Let $H = \{(S_t, T_t)\}$ be the set of every possible cluster, where S_t and T_t are respectively, the set of left nodes (services) and right nodes (clients) belonging to cluster t ($S_t \subseteq S$) and ($T_t \subseteq T$).

t represents the characteristic vector of a cluster, that is : $t_i = 1$, if $i \in S_t$, and $t_i = 0$ otherwise.

Let $c_t = |(S_t \times T_t) \bigcap \bar{E}|$, denote the penalty of cluster t and $a_{it} = 1$, if $i \in S_t,\ t \in \{1, ..., |H|\}$.

We define the following variable:

$$\lambda_t = \begin{cases} 1 & \text{if the cluster } t \text{ is selected.} \\ 0 & \text{Otherwise} \end{cases} \tag{3.29}$$

The linear formulation for the column generation is :

$$\min \sum_{t \in H} c_t \lambda_t \tag{3.30}$$

with constraints:

$$\sum_{t \in H} a_{it} \lambda_t \ = 1 \qquad \forall\ i \in S \tag{3.31}$$

$$\sum_{t \in H} \lambda_t \ = \mathrm{K} \tag{3.32}$$

$$\lambda_t \in \{0,1\} \qquad \forall\ t \in H \tag{3.33}$$

Constraint (3.31) expresses the partition of the services. Constraint (3.32) ensures that the number of clusters used is exactly K. Constraint (3.33) presents the binary variable of the problem, (see Magni *et al.* [66] and Gualandi *et al.* [80,81]).

3.4.5 Fifth formulation

This model is based on integer variables and the clusters are identified by a number from 1 to k. This model suffers from symmetrical solutions. Furthermore it is the most difficult to linearize. Therefore, to linearize this mathematical model, so-called "*ad-hoc*" techniques must be used to reduce this problem to a formulation similar to (third model). Here, x_i is equal to the number of the cluster to which service i belongs and the binary variable z_{ij} is the following:

$$z_{ij} = \begin{cases} 0 & \text{if the edge } (i,j) \text{ is added} \\ 1 & \text{Otherwise} \end{cases} \tag{3.34}$$

This model use the mathematical way to test whether two services (i and j) are in the same cluster k doing by taking the difference of the relative x variables ($x_i - x_j$). If the difference is zero, then, both services i and j share the same cluster. This formulation can be seen as the following:

$$\min \sum_{(i,j)\in\bar{E}} (1 - z_{ij}) \tag{3.35}$$

with constraints:

$$z_{ij} \leq \mid x_i - x_l \mid \qquad \forall\ (i,j) \in \bar{E},\ (\mathrm{l},\ \mathrm{j}) \in E \tag{3.36}$$

$$x_i \in \{1, ..., k\} \qquad \forall\ \mathrm{i} \in S \tag{3.37}$$

$$z_{ij} \in \{0,1\} \qquad \forall\ \mathrm{i} \in S,\ \ j \in T \tag{3.38}$$

Constraint (3.36) forces z_{ij} to take value 0 when the edge (i,j) is added to form the *biclique*. Constraint (3.37) restricts the domain of variable x inside the range of clusters [1 to k]. The constraint (3.38) presents the binary variable z_{ij}, (see Magni *et al.* [66] and Gualandi *et al.* [80,81]).

3.5 Conclusion

In this chapter, we have first recalled some notions on the graphs theory in order to lay the foundations for the problems addressed in this first part of the thesis.

Then, we gave an overview of the problem K-clusters in a bipartite graph. This allowed us to detail some aspects of this problem and especially to introduce the approach that we propose to solve this problem approximately.

The second section of this chapter allowed us, on the one hand, to cite some research works of the literature which treated the problem of K-clusters in a bipartite graph and on the other hand, to cite applications of K -CMBCP in the industrial world.

In the last section of this chapter, we are interested in the main mathematical models. Thus, we note that these models represent a basis for some important characteristics in the literature (see Magni *et al.* [66]) and which we have chosen to enumerate to conclude this chapter:

- Property 1.The optimal solution always uses all the K clusters.
- Property 2. The total penalty for overlapping K-clusters[2] is greater than or equal to those K-clusters that do not overlap.
- Property 3. The continuous relaxation of the first mathematical model generates a solution.
- Property 4. The solution of the first model is integral, when the variables y_{jk} and z_{ijk} are continuous over the interval [0 1].
- Property 5. The solution of the fifth mathematical model is completed, the variable z_{ij} is continuous over the interval [0 1].

In the next chapter, we will propose a variable neighborhood search-based heuristic to approximately solving the problem of K-clusters in a bipartite graph.

[2] It is said that two clusters overlap, when they have a share of common services

Chapter 4

Chapter 4

Variable Neighborhood Search-based Heuristic for the problem of K-clusters in a bipartite graph

Contents

This chapter presents the method of a variable neighbourhood search-based heuristic for approximately solving the K-clustering minimum biclique completion problem. The K-CmBCP consists in partitioning a bipartite undirected graph into K clusters such that the sum of the edges that complete each cluster into a biclique is minimum.

The proposed method is mainly based on three complementary phases. The first phase serves to distribute the services of a bipartite graph to their clusters by sequentially, using a greedy algorithm in order to build a starting solutions.

The second phase, a local search heuristic is performed to improve the quality of the starting solutions. The third and last phase react by considering both diversification and intensification strategies.

The proposed method is evaluated on the set of the standard benchmark instances of the literature. The obtained results are compared with the best results of the literature and the results reached by the Cplex 12.6 solver [41] for a linear mathematical model based on the formulation presented in the previous chapter.

4.1 Introduction

The instance of the K-CmBCP is defined by a bipartite graph $G(V, E)$, where V is the set of vertices such that: $V = S \cup T$ and $S \cap T = \emptyset$ and E represents the set of edges. We also remind that if (S, T) is a biclique, then each vertex of S (respectively of T) is connected to all vertices of T (respectively of S). Therefore, if the graph consists of K clusters: $\Big((S_1, T_1), (S_2, T_2), \ldots, (S_k, T_k)\Big)$, then all the vertices of each pair (S_k, T_k) are interconnected $\forall\ k = \{1, \ldots, K\}$.

Since the bipartite graph $G(V, E)$ under consideration is undirected, search K-clusters is to look for K bipartite sub graph of G with the addition of a minimum of edges that do not belong to E. Similarly, we can say that the goal of the problem is to minimize the total number of missing edges needed to make each cluster is completed by find the best partition of the set S into K clusters.
As described in Gualandi *et al.* [81], K-CmBCP has many applications such as channel partitioning for multi-broadcast transmissions. In this application, given a set of services requested by the clients, the objective is to find K-multicast sessions that partition the set of demands. Consequently, each service should be belonged to one of multicast session while, each client can appear in more than one session [54].
The outline of this chapter is organized as follows: in section 4.2.2, we will present an example of 2-CmBCP and presents these integer linear programming formulations of problem that we solve in the experimental part using the Cplex solver. Section 4.3 will explain the detail of our proposed heuristics for K-CmBCP. Thus, in this section, we will first start by constructing the starting solution using a greedy procedure that builds a solution sequentially, then we will improve this solution by using intensification strategies that include both local search and a neighbourhood search. Finally, we will analyse the performance and results of our approach by comparing it with the best results of the literature and also with the results of the Cplex solver when it is provided the ILP model (see equations (4.2) to (4.7)). The final section concludes this chapter by summarizing the main contribution of this approach to solve the K-CmBCP.

4.2 The mathematical model used

In this section, we first construct an example of two feasible solutions for a K-CmBCP, then we present the mathematical formulation of integer linear programming (ILPM) which will be solved in the experimental part using the Cplex 12.6 solver.

4.2.1 Example with two clusters ($K = 2$)

Figure 4.1 shows a bipartite graph representing an instance of the K-CmBCP. So, in this bipartite graph the services (vertices on the left shore) are represented by the set $\{i_1, i_2, i_3 \ and \ i_4\}$ and the clients (vertices on the right shore) are represented by the set $\{j_1, j_2, j_3, j_4, j_5 \ and \ j_6\}$.

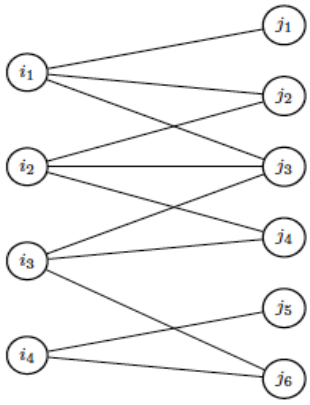

Figure 4.1: Graph representation of the K-CmBCP.

Figure 4.2 provides two possible solutions of the same instance of Figure 4.1 with two bicliques. Such that, the set of services (the left shore vertices):$\{i_1, i_2, i_3 \ and \ i_4\}$ is partitioned into 2- disjoint clusters. Thus, the number of edges to be added to form 2- bicliques is minimum.

In Figure 4.2, the 2-clusters are represented as boxes around the vertices while the additional edges needed to form the two bicliques are represented by dashed arrows and therefore the penalty of each cluster is the sum of its dashed arrows.

The total penalty to the 2-CmBCP for each example of the partitioning (a) and (b) in the Figure 4.2 is the sum of the penalties for all its clusters (see Figure 4.2 (a) and (b)).

4.2.2 The mathematical formulation

In the following, we present a mathematical model from the first model presented in section 3.4.1 of chapter 3, having an integer linear formulation.

Thus, the mathematical model of the problem K-clusters in a bipartite graph be formed as following:

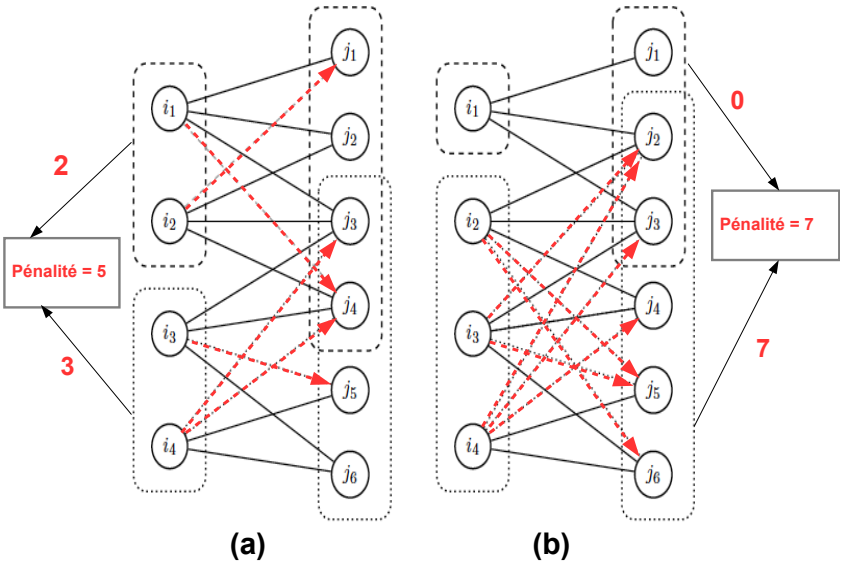

Figure 4.2: Construction of two solutions with two bicliques.

Let the binary variable z_{ijk} be equal to 1 whenever both variables x_{ik} and y_{jk} are equal to 1.

$$z_{ijk} = \begin{cases} 1 & \text{If the nodes } i \text{ and } j \text{ are both assigned to cluster } k \\ 0 & \text{Otherwise} \end{cases} \tag{4.1}$$

Using the linear formulation on the first mathematical model, we obtain the following formulation:

$$(ILPM): \quad \min \sum_{k \in K} \sum_{(i,j) \in \bar{E}} z_{ijk} \tag{4.2}$$

with constraints:

$$x_{ik} + y_{jk} \leq 1 + z_{ijk} \qquad \forall (i,j) \in \bar{E},\ \forall \text{ k} \in \text{K} \tag{4.3}$$

$$\sum_{k \in K} x_{ik} = 1 \qquad \forall \text{ i} \in S \tag{4.4}$$

$$x_{ik} \leq y_{jk}, \qquad \forall \text{ (i, j)} \in E,\ \forall \text{ k} \in K \tag{4.5}$$

$$x_{ik}, y_{jk} \in \{0,1\}, \qquad \forall \text{ i} \in S,\ \forall \text{ j} \in T,\ \forall \text{ k} \in K \tag{4.6}$$

$$z_{ijk} \in \{0,1\}, \qquad \forall\ (i,j) \in \bar{E},\ \forall\ \mathrm{k} \in K \tag{4.7}$$

where :

Constraint 4.2 represents the objective function that minimizes the total penalty of all clusters K. The constraint 4.3 is the constraint that links the variable z_{ijk} to both x_{ik} and y_{jk} variables by setting z_{ijk} to 1 when the other two variables are equal to 1. The constraint 4.4 is the constraint for assigning each service $i \in S$ to a single cluster. The constraint 4.5 forces each client $j \in T$ assigned to the cluster k to be in the same cluster as a service $i \in S$ when j is adjacent to i in E. Finally, constraints 4.6 and 4.7 are binary variables of the problem.

4.3 Neighborhood search-based heuristic for the problem (K-CmBCP)

The main goal of this work is to develop effective methods allowing to solve the problem of K-CmBCP. In this section, we propose approximately solving to the K-CmBCP by using a neighbourhood search heuristic. In this work we propose and implemented three approaches, a heuristic approach based on the principle of cooperation between greedy search, intensification and diversification search. This approach uses three complementary phases:

- **Phase 1:** The first phase builds a starting solution by sequentially using a greedy strategy.
- **Phase 2:** The second phase tries to improve the quality of the starting solution using an *intensification strategy*.
- **Phase 3:** The third and final phase introduces *diversification* in order to achieve certain subspaces in an efficient way. *Diversification* of solutions is considered by applying a process based on the neighborhood search heuristic to improve the feasible solutions, using the perturbation method that consists of two strategies: *degradation* and *reconstruction* strategies.

4.3.1 Construction of a starting solution for K-CmBCP

There are several ways to define a starting solution, but, we proposed the greedy procedure to partition the set of services S of the bipartite graph G and distribute it to their clusters sequentially. Then, in order to provide the penalty of the starting solution reached by the greedy procedure, we use some of computational steps in order to complete each created cluster with the additional edges that forms the K bicliques for G. The sequential method can be described in algorithm 7 which summarizes the construction of a starting solution.

In order to improve the quality of solutions, several methods can be used. In our case, we settle for a sequential scan of the original problem.

We note, also, that Cplex solver can be used to obtain a starting solution. From a classification of services, according to a degree of each service, allocation of services to clusters is determined. This award carries a feasible solution for the K-CmBC problem, where the content of all clusters is assumed balanced.

As can be seen, the quality of the solution depends on the classification, but the classification is performed before resolution and does not impact the algorithm process.

In our algorithm below, we focus on the sequence order of each cluster for distributing all the services to their clusters.

Algorithm 7 Sequential procedure for K-CmBC Problem

Input: K-CmBC instance.
Output: Starting solution (Sol).

```
1: s ← 1;
2: while (each service s ≤ N) do
3:    k ← 1;
4:    for (each cluster k ≤ K) do
5:       Assign the service s to the cluster k
6:       Increment s
7:       if (s > N) then
8:          Exit
9:       end if
10:   end for
11: end while
```

As we see above, the algorithm 7 builds the set of service nodes represented by the index set $S = \{1, ..., N\}$, where N represents the number of service in a bipartite graph and $k = \{1, ..., K\}$, be the set of clusters indices, where K is the number of cluster in a bipartite graph. The solution of the problem starts from the empty set. In this procedure we begin with the first cluster k_1 to assign the first service s_1, then, we assign the second service s_2 to the second cluster k_2 and so on until we reach the last one K.

Similarly, this process can be repeated from the (step 2). For example if we have five cluster ($K = 5$), the service s_6 must be assigned to the first cluster k_1, while, the service s_7 will be assigned to the second cluster k_2 and so on, until the stopping constraint of this procedure is performed, when all services S are distributed to their clusters (see step 6).
A starting solution must be generated in order to even start a local search. It is important that the method used is fast to execute and easy to implement. In fact, sequential procedure is usually extremely fast, yields the same solution at every execution on the same instance. That is different from some methods that generate several solutions randomly. This may be yielded a different solutions at every execution (see [29], [71]).

4.3.2 A local search heuristic to improve the quality of a starting solution.

To improve the quality of a starting solution, there are some procedures are introduced such as, the local search based on number of movements for some services between clusters (k-optimization). Where its aim is to perform an interesting search on a series of neighbourhoods. In our procedure, the substitution among some vertices of services is employed between two different clusters. Thus, we use two optimization to improve the quality of the starting solution. Similarly, these interchanges can be generalized to an σ -neighbourhoods, where σ is a subset of adjacent services, which can be displaced by group to other clusters. From the best solution obtained in these neighborhoods, this process can be repeated. It stops after exploring a number of neighbourhoods (or number of experimentally defined iterations) [61]. We describe in the following the procedure that we use to improve the starting solution.

Algorithm 8 Neighborhood search to improve a starting solution.

Input: Starting solution (Sol).
Output: Feasible solution (Sol').

1: **while** (stopping criterion is not performed) **do**
2: `Procedure` –Swap Two Cluster.
3: Sort all the clusters K in *decreasing* order.
4: Sort the services of each cluster in *decreasing* order.
5: Set $k_1 \leftarrow 1$, where $k_1 \in \{1, ..., K\}$
6: **for** (each services $i = \{1...S1\}$), where $S1$ denotes the services of k_1 **do**
7: **for** (each cluster $q = \{K...k_2\}$), where $q \neq k_1$ **do**
8: **for** (each services $j = \{S2...1\}$), where $S2$ denotes the services of q **do**
9: Swap the service i with the service j.
10: Find new solution(Sol').
11: **if** ($Val(Sol') \leq Val(Sol)$) **then**
12: $Sol \leftarrow Sol'$.
13: *Resort* the services of clusters (k_1, q).
14: **else**
15: Search in neighborhood of the service j.
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: `end procedure`
21: **end while**

Algorithm 8 describes the main steps of the intensification procedure which is used for improving the quality of the solutions at hand. Indeed, the input of the algorithm is the solution provided by the *sequential procedure*. The first loop in (line 1) is used to stop the resolution when the local search is able to find a feasible solution with a lower penalty. The stop criterion is also consolidated

with the limit number of iterations that frame exploration of the neighbourhood. As we can see, the classification and selection process are performed for all clusters and their services before any movement for services between two selection clusters. As can be seen in (line 3, 4), in our algorithm, the classifications and selection process are based on the number of the missing edges added to complete each cluster (i.e. to become a biclique). Then, an exchanging steps are considered as follows:

1. A vertex i belonging to a cluster k_1 can be exchanged with another vertex j that belongs to another cluster q (see line 9), the solution is updated when the new solution improves the quality of the current solution of the neighborhood. Then, we will re-sort the set of the services of clusters k_1 and q in decreasing order based the number of missing edges (see line 13).
2. When the previous step fails to provide a better solution then, the search process will be applied with the other neighborhoods services that have up degree of the service j, (see line 8, 15). This process can be repeated, it stops whenever the local search is able to provide a new feasible solution.

   In the VNS search, the services can be moved into another cluster are not limited, and are based on the solutions obtained by moving one service to another cluster at a time.

   It is important that the above proposed algorithm is very fast and efficient for improving the starting solution and producing the best feasible solution.

4.3.3 Perturbation method for solutions diversification

In a search process, we often find treatments that serve to diversify solutions. This diversification aims to explore other research sub-areas, which probably have not been visited. Among these diversification processes, there are procedures based on disturbances of solutions. In our study, we were more interested in the disruption of solutions, which is to disturb the neighborhood of current solutions. In this work, we can improve the feasible solution by using the perturbation method that combines two strategies: *degrading* and *re-constructing strategies* (as used in Hifi and Michrafy [52]). After applying both strategies, the intensification step (algorithm 8) is recalled in order to improve the quality of each solution at hand.

In the following, we will show how to combine both strategies for searching a series of new solutions. Let (Sol') be the current feasible solution obtained from algorithm 8. Let α be a constant, such that $\alpha \in [0, 100]$, denoting the percentage of free vertices of the set S, i.e., some vertices are free according to the current solution Sol'. Then, the diversification procedure can be considered as an alternate approach which is performed by using two procedures that are:

- The *degrading strategy:* is used to destroy $\alpha\%$ elements randomly from the current feasible solution for extracting a partial solution P.

- The *re-constructing strategy:* is used to complete the partial solution obtained after destruction and re-optimize the complementary solution $P^{'}$ by application the Cplex solver and to improve the current solution. This procedure is based on destroying a part of the current solution and substituting with the elements of its neighbours.

Algorithm 9 Destroy and repair process

Input: Feasible solution (Sol') obtained by VNS, Max: Maximum iteration number.
Output: An approximate solution Sol_{global}.

1: $Sol_{best} \leftarrow Sol'$.
2: **while** (stopping criterion is not performed) **do**
3: $Sol_{global} \leftarrow Sol_{best}$.
4: **while** ($LocalIteration < Max$) **do**
5: $Sol_{local} \leftarrow Sol_{best}$.
6: *Destroy($\alpha\%$)* from (Sol_{local}, call degrading procedure in order to provide a partial solution P.
7: *Repair(P)*, re-optimize the complementary solution $P^{'}$ for realizing a new solution ($Sol_{current}$).
8: **if** ($Sol_{current} < Sol_{best}$) **then**
9: $Sol_{best} \leftarrow Sol_{current}$.
10: **end if**
11: **end while**
12: **end while**

the details of this procedure presented in algorithm 9 that started by consider the current solution obtained by algorithm 8 as a best solution and it is setting to Sol_{best} in (line 1) as a global solution we want to optimize by combining both degrading and re-optimizing strategies. The first loop in (line 2) is used to stop the search process, when the stopping criteria is performed; herein, a maximum number of iterations is considered. The second loop in (line 4) serves as local loop for search the best solution on a series of neighborhoods. Next, the degrading procedure is called in (line 6) a local loop for randomly removing $\alpha\%$ of the elements from the current solution and trying to improve the quality of solution by the re-optimization procedure in (line 7). Then, the best solution is compared with the current solution for extracting the best feasible solution. Similarly, this approach can be repeated until a maximum number of diversification or limiting overall execution time for the judgement of the method.

4.4 Computational results

In this section, the most significant results are presented. Our goal is to show and discuss the results of the proposed algorithms in section 4.3 that effectively allows to approximately solving the K-CmBCP. Our computational results are presented in series of three parts:
The first part presented in table 4.2 shows solutions of sequential procedure presented in algorithm 7. The results of second part presented in tables 4.2 and 4.5 show the solutions of neighborhood search presented in algorithm 8. The third part presented in tables 4.3 and 4.6 investigates effect of destroying process on the quality of the final solution produced by perturbation search presented in algorithm 9.

4.4.1 Description of the instances of K-CmBCP

Two types of instances can be considered: *symmetric* instances, where the number of services $|S|$ and the number of clients $|T|$ are equal and *non-symmetric* instances where sometimes the number of clients is greater than the number of services (or vice versa).
Note that, in this part of thesis, we considered *symmetric* instances. The table 4.1 describes the main characteristics of the instances tested that extracted by Gualandi *et al.* [81]. Column 1 indicates the label of the instance, the columns from 2 to 4 respectively display the number of services $|S|$, the number of clients $|T|$ and the number of clusters $|K|$ and finally, column 5 tallies the density d of the associated bipartite graph for each instance.
Note also, the K-CmBCP instances are composed of several subgroups according to the variation of the number of services $|S|$, number of clients $|T|$, the number of clusters $|K|$ and the density d ($d = 0.3$, $d = 0.5$, $d = 0.7$) of its corresponding graph.

4.4.2 Parameter settings

The numerical results obtained by Cplex solver [41] version 12.6 was executed on a multi-users windows laptop (2.8 GHz, 4 Gb), with 3600 seconds cpu time limit, based on the model in section 4.2.2. The proposed algorithms was coded in C++ and executed on the same machine with 100 seconds cpu time limit. So, for test both (Cplex and proposed algorithms), we use the instances of Gualandi *et al.* [81], (see table 4.1).
On the other hand, in the numerical results, there are two main parameters should be taken into account by our proposed method: the stopping criterion and the percentage of the removed vertices belonging to the current solution. In order to maintain the diversity of solutions, we set the local stopping criterion to 20 iterations and the maximum number of global iterations was fixed to 50.

Inst.	S	T	k	d
I1.1	15	15	2	0.3
I1.2	15	15	2	0.5
I1.3	15	15	2	0.7
I2.1	15	15	5	0.3
I2.2	15	15	5	0.5
I2.3	15	15	5	0.7
I3.1	18	18	2	0.3
I3.2	18	18	2	0.5
I3.3	18	18	2	0.7
I4.1	18	18	5	0.3
I4.2	18	18	5	0.5
I4.3	18	18	5	0.7
I5.1	20	20	5	0.3
I5.2	20	20	5	0.5
I5.3	20	20	5	0.7
I6.1	50	50	5	0.3
I6.2	50	50	5	0.5
I6.3	50	50	5	0.7
I7.1	50	50	10	0.3
I7.2	50	50	10	0.5
I7.3	50	50	10	0.7
I8.1	80	80	5	0.3
I8.2	80	80	5	0.5
I8.3	80	80	5	0.7
I9.1	80	80	10	0.3
I9.2	80	80	10	0.5
I9.3	80	80	10	0.7
I10.1	100	100	5	0.3
I10.2	100	100	5	0.5
I10.3	100	100	5	0.7
I11.1	100	100	10	0.3
I11.2	100	100	10	0.5
I11.3	100	100	10	0.7

Table 4.1: The description of instances for the K-CmBCP.

In our numerical results, four values of the parameter α were considered: $\alpha \in \{2\%;\ 5\%;\ 10\%;\ 15\%\}$.

4.4.3 Test the performance of the proposed methods

Table 4.2 summarizes the results obtained in initialization mode with our sequential approach (algorithm 7), ILPM Cplex solver and with our neighborhood search algorithm. Table 4.2 compares the average performance of these three methods.
Column 1 displays the name of each instance. The objective values of the best

solutions are presented in column 2. Columns 3 and 4 display the solutions and run time of the sequential algorithm for each instance. Column 5 reports the objective value of the best solutions provided by Cplex solver within 3600 seconds. Columns 6 and 7 show respectively the objective values and the real run time realized by our variable neighborhood search based on a starting solution provided by sequential algorithm. The last two rows show the averages values for all solutions and their cpu times.

Inst.	*Best*	*SEQUENTIAL*		*ILPM*		*VNS*
		Sol.	*Cpu.*	*Cplex*	*Sol.*	*Cpu.*
I1.1	115	156	0.03	115	123	0.01
I1.2	93	113	0.01	93	95	0.01
I1.3	63	71	0.01	63	63	0.01
I2.1	52	90	0.01	52	71	0.02
I2.2	51	80	0.001	51	62	0.02
I2.3	41	65	0.001	41	43	0.02
I3.1	180	204	0.001	180	190	0.02
I3.2	127	156	0.001	127	136	0.02
I3.3	89	99	0.001	89	89	0.02
I4.1	94	138	0.001	94	126	0.04
I4.2	87	137	0.001	87	96	0.04
I4.3	63	96	0.01	63	65	0.04
I5.1	120	179	0.03	120	174	0.05
I5.2	124	164	0.001	124	132	0.05
I5.3	90	128	0.01	92	90	0.05
I6.1	1484	1651	0.03	1485	1484	0.46
I6.2	1092	1244	0.01	1100	1092	0.5
I6.3	675	761	0.02	685	675	0.47
I7.1	1151	1316	0.02	1151	1201	1.05
I7.2	924	1154	0.01	936	924	1.15
I7.3	582	746	0.01	588	582	1.11
Avg Sol.		***416.57***		***349.33***	***357.76***	
Avg cpu.			***0.01***	***3600.00***		***0.26***

Table 4.2: Performance of sequential algorithm, Cplex solver and neighborhood search with single iteration

The analysis of table 4.2 is as follows:

1. We recall that our problem is a minimization problem. In the context of the quality of solutions, the average value produced by Cplex solver are better than the average values produced by other approaches, though, the resolution time is 3600 seconds for each instances. While, we can observe the acceleration of the average runtime of the sequential procedure to 0.01 seconds and the average runtime of variable neighborhood search to 0.26 seconds.

2. We note that neighborhood search actually improves the performance of

sequential resolution and this from the first iteration. The average solutions of the neighborhood search approach decreases at 14% compared with the average solutions of sequential algorithm.

3. One can observe that neighborhood search algorithm is able to reach and improves the best solution provides by Cplex solver for some instances. In fact, VNS realizes 8 best solutions out of 21.

4.4.4 Effect of the destroy and repair process

This section evaluates the effect of degradation and reconstruction procedures on our approach. Recall that the proposed approach works as follows: First, the current feasible solution is degraded by removing α% from the fixed variables. Then, the partial solution obtained is reconstructed using the greedy procedure and enhanced using the Cplex solver and intensification phase. Finally, the same resolution on the last solution until there is no better solution or until the maximum number of iterations is reached.

Table 4.3 summarizes the results obtained when we vary the destroy percentage. Column 3 shows the quality of the final solution when the random destruction percentage is 2%, column 4 shows the quality of the final solution when the random destruction percentage is 5%, column 5 shows the quality of the final solution when the random destruction percentage is 10% and finally, column 6 shows the quality of the final solution when the random destruction percentage is 15%.
The analysis of table 4.3 shows that destroy of 2% of the previous iteration solution seems the most effective setting.

Table 4.4 shows comparison of some results realized by VNS and those extracted by the best method of the literature Gualandi *et al.* [81] (noted GMM). And also we can observe the solutions obtained by the mathematical model ILPM (see section 4.2.2), when solved using the Cplex 12.6 solver (note that we have limited the resolution time of the Cplex solver to one hour). Column 1 of this table represents the label of the instance, Column 2 and 3 display the best solutions and resolution times provided by VNS, Column 4 shows the solutions obtained by the Cplex solver on the mathematical model ILPM, Column 5 and 6 display the best solutions of the literature (Gualandi *et al.* [81]).

The analysis of table 4.4 shows that:

1. We can observe the inferiority of Cplex solver since it is not able to match the best solutions of the literature.

2. The best solutions realize by VNS are better than the solutions reached by cplex solver. Indeed, VNS is able to realize or approximate the solutions of the literature with minimum run time.

Inst.	*Best*	$\alpha\%$			
		2%	5%	10%	15%
I1.1	115	115	115	115	115
I1.2	92	93	93	92	92
I1.3	63	63	63	63	63
I2.1	52	52	52	52	52
I2.2	51	51	51	51	51
I2.3	40	40	40	40	40
I3.1	179	180	179	180	184
I3.2	127	127	127	127	127
I3.3	89	89	89	89	89
I4.1	94	94	94	94	94
I4.2	87	87	87	87	88
I4.3	63	63	63	63	64
I5.1	120	120	120	121	120
I5.2	123	123	126	123	123
I5.3	82	82	82	82	82
I6.1	1401	1401	1418	1425	1467
I6.2	1090	1090	1094	1100	1092
I6.3	675	675	676	676	676
I7.1	1110	1137	1110	1196	1191
I7.2	920	920	922	922	920
I7.3	582	582	584	591	596
AV Sol.	***340.7***	***342.10***	***342.14***	***347.09***	***348.71***
AV cpu.		***100.00***	***100.00***	***100.00***	***100.00***

Table 4.3: Performance of perturbation method with the random selection policy and 100 seconds cpu time limit

4.4.5 Test the performance of proposed method with the large sizes instances:

Table 4.5 summarizes the results obtained when we apply sequential algorithm, ILPM Cplex solver and our neighborhood search algorithm on large instances extracted by Gualandi *et al.* [81].

Columns 3 and 4 respectively give solutions and resolution times obtained by sequential algorithm, column 5 gives the results obtained by the Cplex solver within 14400 seconds, columns 6 and 7 respectively give solutions and resolution times obtained by neighborhood search algorithm.

Table 4.6 shows the solutions found when applies perturbation method to large instances: column 2 gives solutions obtained when the random destruction percentage is $\alpha = 2\%$, column 3 gives solutions obtained when the random destruction percentage is $\alpha = 5\%$ and column 4 gives the solutions obtained when the random destruction percentage is $\alpha = 10\%$ and column 5 gives the solutions

Inst.	UB_{VNS}	CPU_{VNS}	$ILPM_{cplex}$	UB_{GMM}	CPU_{GMM}
I2.1	52	0.13	52	51	8.11
I2.2	51	0.121	51	50	7.91
I2.3	40	0.121	41	39	7.17
I4.1	94	0.141	94	92	40.17
I4.2	87	0.141	87	86	58.02
I4.3	63	0.15	63	63	29.06
I5.1	120	0.18	120	119	183.32
I5.2	123	0.151	124	123	119.09
I5.3	82	0.16	92	82	131.31
I6.1	1401	0.59	1485	1321	107
I6.2	1090	0.61	1100	1072	11
I6.3	675	0.59	685	672	16
I7.1	1110	1.17	1151	938	137
I7.2	920	1.26	936	876	50
I7.3	582	1.22	588	577	43
Avg.	***432.66***	***0.4389***	***444.6***	***410.73***	***63.2106***

Table 4.4: Comparison of the some solutions that are obtained with the ILPM and literatures solutions

obtained when the random destruction percentage is $\alpha = 15\%$. We can compare our results presented in column 6 with those realized by the literature and with the results obtained by the Cplex solver in columns $7, 9$.

Inst.	*Best*	*SEQUENTIAL*		*ILPM*	*VNS*	
		Sol.	*Cpu.*	*Cplex*	*Sol.*	*Cpu.*
I6.1	1423	1651	0.03	1423	1484	0.58
I6.2	1091	1244	0.01	1091	1092	0.62
I6.3	675	761	0.02	676	675	0.62
I7.1	1135	71316	0.02	1135	1201	1.28
I7.2	924	1154	0.01	930	924	1.40
I7.3	582	746	0.01	587	582	1.64
I8.1	4037	4459	0.022	4166	4037	2.37
I8.2	2880	3132	0.026	2931	2880	2.75
I8.3	1771	1901	0.028	1775	1771	2.62
I9.1	3601	4123	0.025	3735	3601	6.26
I9.2	2618	3116	0.026	2675	2618	6.90
I9.3	1619	1901	0.028	1634	1619	7.42
I10.1	6484	7034	0.030	6645	6484	4.94
I10.2	4682	4968	0.031	4716	4682	5.49
I10.3	2842	3003	0.035	2854	2842	4.93
I11.1	5874	6734	0.031	6119	5874	12.90
I11.2	8812	9990	0.038	9111	8812	27.10
I11.3	2654	3003	0.040	2704	2654	14.75
Avg.		3346.44	0.04	3050.39	2990.67	5.81

Table 4.5: Performance of sequential algorithm, ILPM Cplex solver and neighborhood search with large sizes instances

	Variation of α for large instances vs Cplex and literature.							
#Inst	$UB_{2\%}$	$UB_{5\%}$	$UB_{10\%}$	$UB_{15\%}$	UB_{Best}	UB_{GMM}	CPU_{GMM}	UB_{Cplex}
I6.1	1401	1418	1425	1467	1401	1321	107	1423
I6.2	1090	1094	1100	1092	1090	1072	11	1091
I6.3	675	676	676	676	675	672	16	676
I7.1	1137	1110	1196	1191	1110	938	137	1135
I7.2	920	922	922	920	920	876	50	930
I7.3	582	584	591	596	582	577	43	587
I8.1	4037	4139	4037	4035	4035	3819	872	4166
I8.2	2864	2880	2880	2880	2864	2862	167	2931
I8.3	1771	1782	1771	1771	1771	1769	46	1775
I9.1	3587	3733	3739	3739	3587	3202	780	3735
I9.2	2612	2739	2785	2785	2612	2571	400	2675
I9.3	1618	1646	1696	1619	1618	1618	144	1634
I10.1	6484	6513	6513	6513	6484	6248	452	6645
I10.2	4682	4702	4684	4684	4682	4658	198	4716
I10.3	2842	2842	2842	2842	2842	2842	87	2854
I11.1	5874	5921	4298	4298	5874	4298	893	6119
I11.2	8788	8796	8796	8796	8788	5428	5427	9111
I11.3	2654	2872	2654	2654	2654	2644	441	2704
Avg.	**2978.78**	3020.50	3012.67	3010.06	**2977.17**	**2634.17**	570.61	**3050.39**

Table 4.6: Comparison of some numerical results obtained by perturbation method with the ILPM and literatures solutions

The analysis of tables (4.5 and 4.6) shows that:

1. The solutions provided by our approach VNS in table 4.5 are better than those obtained by the Cplex 12.6 solver (see column 5 and Column 6).

2. The resolution times that spend by our VNS are very small vs those resolution times spent by the Cplex 12.6 solver (see column 7).

3. The encouraging results of neighborhood search, push us to go further in setting the resolution parameters. In fact, the previous results are relative to a destroy of 2% of the feasible solution. We studied as in the following, the performance of perturbation search when we vary percentage of items to be destroyed from 2% to 15%, staying on a random selection policy with 100 seconds cpu time limit. These results are shown in the table 4.3 and table 4.6. The effect of the variation of α, concerning, the solution on large instances of table 4.6 is almost the same as on the small instances of table 4.3. It is also noted that, for all large instances that we have tested, the solutions obtained with $\alpha = 2\%$ are better than or match to those obtained with $\alpha = 5\%$, $\alpha = 10\%$ and $\alpha = 15\%$.

   However, we can easily observe that, overall, the quality of the solutions obtained for large instances, is better when, we fix α to 2%.

4.5 Conclusion

In this chapter we proposed a new variable neighborhood search-based heuristic for approximately solving to the K-clustering minimum bi-clique completion problem. This approach is based on some procedures: a greedy procedure that used for partition the services according to their clusters, hybrid method that combines the neighbourhood search based heuristic and perturbation procedure both serves to intensify and diversify the search process. Our approach uses phases which is a local search that will try to improve the quality of a series of solutions.

The numerical results showed that the proposed approach provides encouraging results by improving the quality of solutions for known instances of the literature (Gualandi *et al.* [81] and the quality of solutions provided by the Cplex.12.6 solver on the mathematical model used (see section 4.2.2) for large size instances.

The heuristic was published in an international conference with proceedings 3rd IEEE International Conference on Control, Decision and Information Technologies, CoDIT 2016 (Hifi *et al.*. [54]). And was presented during the 17th and 18th Conference of the French Society for Operational Research and Decision, ROADEF 2016 [55], ROADEF 2017 [56].

Chapter 5

Part II

The Quadratic Knapsack Problem(QKP)

Chapter 5

Quadratic knapsack problems and resolution methods

Contents

In this chapter we first recall the state of the art of some combinatorial optimization problems belonging to the quadratic knapsack family. Next, we present some notions about linear programming. Moreover, we discuss several resolution methods to solve the combinatorial optimization problems which can be divided into two categories: The exact methods which promise the optimality of the solution and the approximate methods such as: hybrid, heuristics and meta-heuristic methods produce an approximate solutions.

5.1 Introduction

Knapsack problems (namely KPs)have been studied for more than a century, that is because they have widely significant applications in numerous domains, particularly in the transport logistics, industry, financial management [33]. Quadratic knapsack problems(QKP) contain a family of knapsack problems with quadratic objective functions. The QKP is a well-studied combinatorial optimisation problem and has been shown strongly NP-hard. This problem has been applied with a variety of important applications, such as, in the location of satellites, airports, railway stations or freight terminals. In the literature, there are different kinds of problems of knapsack model like (QKP). A variety of resolution methods are available to solve it also such as: exact and approximate, heuristics, meta-heuristic and hybrid. Although QKP has not been studied as intensively, the related numerous papers dealing with the problem have been introduced during the last years. In 1980, Gallo *et al.* [28] invented the QKP and presented a family

of upper bounds based on upper planes, which are linear functions of the binary variables satisfying that their value is not smaller than the QKP objective function over the set of feasible QKP solutions. Johnson *et al.* [48] considered the graph version of the QKP. After linearization of the objective function, the model is solved by a branch-and-cut system in which tree inequalities and star inequalities are used to tighten the formulation. Billionnet and Calmels [4] used a classical linearization technique of the objective function to obtain an ILP formulation. As the linearized model may grow quite large, a delayed formulation method is used in a branch-and-cut manner. Lagrangian relaxation approaches are described by Chaillou *et al.* [60]. Relaxing the capacity constraint, a quadratic optimization problem appears which is solvable in polynomial time through a maximum flow problem. Michelon and Veilleux [63] used a Lagrangian decomposition technique to split the problem into a quadratic 0-1 optimization problem and a KP. Létocart and Nagih [49] introduced reoptimization in Lagrangian methods for the 0-1 quadratic knapsack problem. The quadratic optimization problem has some suitable properties, which makes it solvable by use of the techniques introduced by Chaillou *et al.*. Hammer and Rader [70] used the upper bound by Chaillou *et al.* in their computational study, but improved the algorithm by using order relations to fix variables inside a branch-and-bound algorithm. Hammer and Rader also presented an improved heuristic based on the best linear approximation of QKP. Helmberg *et al.* [10] consider a more general version of the problem where P may have negative entries. Several upper bounds are presented based on a cascade of semidefinite programming relaxations. To strengthen the formulation, a number of valid inequalities are derived based on the ordinary KP polyhedron, as well as specific inequalities for the QKP polyhedron. Caprara *et al.* [5] used Lagrangian relaxation of the symmetry constraint $x_i x_j = x_j x_i$ to reach a reformulation of the problem through sub-gradient optimization. Using the reformulated problem, upper bounds tighter than those presented by Gallo, Hammer, Simeone can be derived in $O(n)$ expected time inside a branch-and-bound algorithm. Billionnet *et al.* [2] presented a bound based on the partitioning of N into m disjoint classes. Using Lagrangian decomposition, the problem can be split into m independent sub-problems, which are easier to solve. Rader and Woeginger [42] finally presented a fully polynomial approximation scheme for a QKP defined on an edge series parallel graph. They also proved that QKP with positive and negative profits does not have any polynomial time approximation algorithm with fixed approximation ratio. A greedy genetic algorithm is proposed by Julstrom [44].
As one might expect, due to its generality, QKP has a wide spectrum of applications. Witzgall [82] presented a problem which arises in telecommunications when a number of sites for satellite stations have to be selected, such that the global traffic between these stations is maximized and a budget constraint is respected. This problem appears to be a QKP. Similar models arise when considering the location of airports, railway stations or freight handling terminals [75]. Johnson *et al.* mention a compiler design problem which may be formulated as a QKP, as described in [10]. Dijkhuizen and Faigle [27] and Park *et al.* [45] consider

the weighted maximum biclique problem. If all edge weights are non-negative, this problem is the special case of QKP arising when $w_j = 1$ for $j \in N$ and $b = c$. Ferreira *et al.* [9] consider a problem in VLSI design where large graphs need to be decomposed into smaller graphs of tractable size. The corresponding optimization problem for a single sub-graph can be recognized as a QKP in the minimization form. On the other hand, QKP is shown as the pricing problem for solving a graph partitioning problem using column generation method as described in Johnson *et al.* [48]. Finally, in 2016, Yuning *et al.* [35] presented the memetic search for the generalized quadratic multiple knapsack problem.

5.2 Definition

In all variants of the quadratic knapsack problems, for a set of given items, profits are not only assigned to individual items but also to pairs of them. The pairwise profit is added to the quadratic objective value only when two corresponding items are included in the same knapsack.
Assume a quadratic knapsack with a fixed capacity c and a set of candidate objects (or items), that a set $N = 1, 2, ..., n$, where each item i posses a positive weight w_i and a profit p_{ij}. The $n \times n$ positive integer matrix $P = P_{ij}$ are given, where $p_{ij} + p_{ji}$ is a profit achieved by selecting two different items i and j. If i is selected, generates an object profit p_{ij} and the pairwise profit p_{ji} which is added to the quadratic objective value only when any other selected object j is included in the same knapsack. In addition, each pair of objects i and j are $(1 \leq i \neq j \leq n)$.
The objective of this problem is to select a subset of objects for including into knapsack, so as to maximize the overall profit P, while the overall weight does not exceed the knapsack capacity c. Assume that $max_{i \in N}\ w_i \leq c < \sum_{i \in N} w_i$. If the first inequality is violated, i.e. if $w_i > c$ for some i, we may fix the decision variable x_i to value 0. If the second inequality is violated a trivial solution exists with all items chosen. The problem is well presented in [14, 24, 34].

5.2.1 Notions about Linear Programming

A mathematical program is said to be presented in the form of a linear program (denoted by LP) when its objective function and its constraints are linear. A linear programming problem consists of minimizing (or maximizing) a linear function under certain linear constraints. Thus, a form of a linear program in the case of a maximizing, is the following:

$$(P) \begin{cases} \max\ f(x) \\ \\ \ Subject\ To: \\ \mathrm{g}_i(\mathrm{x}) \geq 0,\ i = 1, ..., M \\ \mathrm{x} \in (x_1, x_2, ..., x_N) \in R \end{cases} \tag{5.1}$$

Where the functions f and g_i are linear functions with variables x_i,..., x_N. The function f is called objective-function and the conditions represent the constraints of the problem (P). A solution to the problem (P) is a vector that satisfies all constraints. The cost of a solution is the value of the objective function f. An optimal solution of the problem (P) is the solution that maximizes the value of $f(x)$ among all feasible solutions.

On the other hand, it is said that a mathematical program is an integer linear program (noted: ILP) [41] when the values of the variables of the linear program are integers. Note that a mathematical program can also be presented in the form of a mixed linear program (noted MLP). The latter is obtained when some of the variables of the problem are integer and others are not.

5.2.2 0-1 Quadratic knapsack problem (QKP)

The "0-1" quadratic knapsack problem (QKP) can be settled by introducing the binary decision variables x_i for indicating if the item i is selected (see section 5.2), [3, 19, 63].
This problem can be formulated as the following:

$$\max : \quad f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij} x_i x_j \tag{5.2}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_j \leq c \tag{5.3}$$

$$x \in \{0,1\} \qquad \forall i \in N = \{1, ..., n\} \tag{5.4}$$

where x_i is a binary variable, that can be formed as the following:

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is included in the knapsack} \\ 0 & \text{otherwise.} \end{cases}$$

As we can see, there are three equations for formulating the quadratic knapsack problem. The first equation (5.2) is the objective function that maximizes the total profit P_{ij} of items included in the QKP based on two constraints. The first constraint (5.3) is defined as the capacity constraint, which forces that the weights' sum of the chosen items does not exceed the knapsack capacity. While, the second constraint (5.4) gives the binary variable of the problem and expresses the items that are to be included or not in the knapsack (i.e. it does not include any trivial item) where, it is assumed that:

- Specified the positive integers for all input data c, p_{ij}, w_i, $\forall i \in N$.
- $\sum_{i \in N} w_i > c$ avoiding this solutions.

5.3 The classical resolution methods

Methods for solving optimization problems such as quadratic knapsack problem (QKP) are generally divided into two categories: the exact and the approximate methods (heuristics and metaheuristics) designed to find an optimum of a problem. The exact methods often have calculation times that increase exponentially with the size of the problems they are trying to resolve. Therefore, they must be stopped prematurely. In this case, the exact methods produce only minor called lower bounds or major called upper bounds.

The computation of the upper and lower bounds permits you to frame the value of the optimal solution of a problem being addressed (i.e. limits the range of values that contain the optimal solution). Thus, they can give an indication of how far away we are from the optimal solution and how one should proceed to construct an improved solution. Indeed, these bounds can be used for the development of exact solution methods based in general on an enumeration procedure.

The choice of the bounding methods is usually a trade off between the tightness of the bound such obtained and the computation time to achieve it. As concerns, the 0-1 quadratic knapsack problem, the upper bounds produced by the known exact approaches are based on Lagrangian relaxation and decomposition (see [5, 49, 63]).

Conversely, the approximate methods are designed to produce solutions of good quality with a reasonable resolution time, but not necessarily optimal. Thus, the approximate methods include heuristics, local searches that give a sequence of decreasing (or increasing) cost solutions up to the local optimum, and finally metaheuristics that use various mechanisms to get away from local optima.

5.3.1 The exact methods

The basic principle of an exact algorithm is, in general, the enumeration of the set of solutions in the search space implicitly. Often, this type of algorithms solves the small problems only. Otherwise, the computing time increases exponentially with the size of the problem. There are numerous exact methods have been presented to find the optimum solutions of a quadratic knapsack problem such as: branch and bound and dynamic programming (see [1, 2, 5]).

Branch and bound method

The goal of a branch-and-bound algorithm is to find a value x that maximizes or minimizes the value of a real-valued function $f(x)$, called an objective function, among some set S of admissible, or candidate solutions. The set S is called the search space. The concept of branch and bound methods are focused on enumeration of the search space then selecting the best solution to a given problem [46, 47].

The branch and bound algorithm is that recursively splits the search space into

smaller spaces to simplify testing them. Branching is termed to generate new branches in the enumeration tree. To improve the performance of a branch-and-bound, algorithm keeps track of bounds on the minimum that it is trying to find, and uses these bounds to "prune" the search space, eliminating candidate solutions that it can prove will not contain an optimal solution. There are several strategies to choose the next node in the search tree such as: depth first and best first strategies [51]. The branch and bound algorithm for the QKP are proposed and several developments have been introduced later in the literature, for more details see section 5.1.

Dynamic programming

The dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions. This method is invented by Bellman [76], who defined it as both an optimization method that operates in sequences and whose efficiency is based on the principle of Bellman optimality, that an optimal sequence is composed of optimal sub-sequences. Thus, this principle gives a method of ascending resolution that determines the optimal solution to a problem(see [59]). This approach consists in partition of the problem into simpler sub-problems, and solving of all its sub-problems that begin with the smallest sub-problems and then go back to the most difficult sub-problems by taking advantage of the results of the problems already solved and then, collecting the sub-solutions to obtain the overall solution. In 2014 Fomeni *et al.* presented a dynamic programming heuristic for the quadratic knapsack problem (see [21]).

5.3.2 The approximate methods

The approximate methods are considered as an interesting alternative to solve large optimization problems since in this case optimality is not paramount. These methods include: heuristics and metaheuristics represented mainly by neighbourhood methods (local search, simulated annealing, tabu search) and evolutionary algorithms (genetic algorithms, ant colonies, etc.). In this way:

- Empirically, by comparing its result (ie, the computation time and the value of the solution obtained) with the best known solution or with a lower or upper bound (depending on the objective of the problem).
- Mathematically, calculating the ratio in the worst case between the optimum and the value of the solution obtained.

Heuristic methods

A constructive heuristic aims to provide in a polynomial time a good solution to an optimization problem but without guarantee of optimality (see [22]). For

the quadratic knapsack problems, among the heuristic methods, we cite here a method based on the greedy principle proposed by (Dasgupta *et al.* [78]). The greedy methods are a class of heuristic methods that build a solution by focusing entirely on immediate improvement that yields a starting solution for the QKP.

Reactive search

Reactive search finds feasible solutions if it exists, and it is generally very efficient, yielding a best solution in a relatively short amount of time, but the quality of a solution changes a lot based on the problem and the chosen evaluation method used during solving, (see [71, 72]).

Intelligent optimization, a superset of reactive search optimization, refers to a more extended area of research, including online and off-line schemes based on the use of memory, adaptation, incremental development of models experiment [77]. The metaphors for reactive search are derived from the way human brain learns and makes decisions based on previous experience or facts. One of the human learning processes is called "*observational learning*" or famously known as "*learning by example*". Human brain learns by observation and repeats the learned subject by repetition. This is the main inspiration source for inserting machine learning into the optimization engine of reactive search. In searching for solutions, many alternative solutions are tested in the exploration of a search space. Intelligent optimization is a perfect introduction to the essential of reactive search, as well as an effort to develop some fresh intuition for the approaches. There are specific problems have been widely studied by reactive search and heuristics intelligent optimization such as: reacting on the neighbourhood.

In the search space, $x^{(i)}$ the current solution at iteration number i. $N(x^{(i)})$ is the neighborhood of point $x^{(i)}$, obtained by applying a set of basic replacement $\{r_0\ ,\ r_1, ..., r_R\}$ to the current configuration: $N(x^{(i)}) = \{x \in X$ such that $x = r_j(x^{(i)}),\ j = \{0, ..., R\}\}$ local search starts from an admissible configuration $x^{(0)}$ and builds a search trajectory $x^{(0)}, ..., x^{(i+1)}$. The successor of the current point is a point in the neighborhood with an upper value of the function f to be maximized. If no neighbour has this property, i.e., if the configuration is a local maximum, the search stops.

Figure 5.1 illustrates the reactive strategy: point **a** corresponds to local minimum, point **b** is the best point in neighborhood N_1, and point **c** the best point in N_2. The value of point **c** is still worse, but the point is in a different attraction basin so that a better **e** could now be reached by the default local search. The best point **d** in N_3 is already improving on **a**, [77].

Diversifying methods

Another essential heuristic approach is a local search that enhances the quality of the initial solution that is obtained by the greedy algorithm. Diversifying

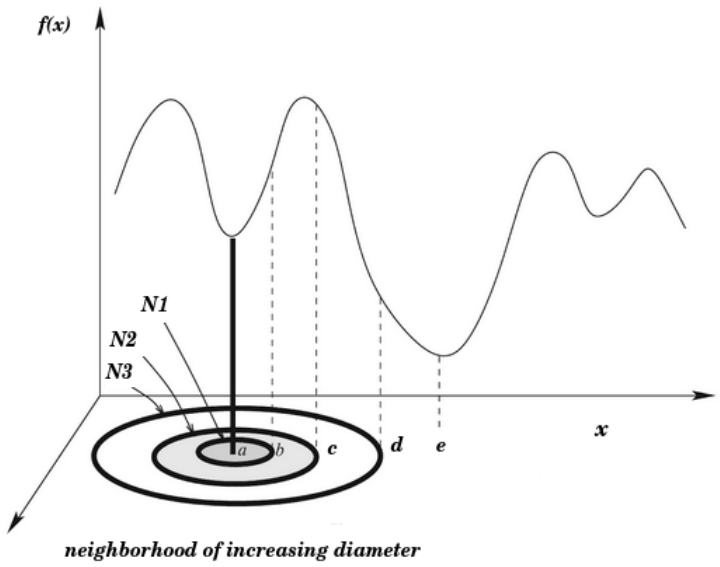

Figure 5.1: An example of variable neighbourhood of different diameters.

methods are research in nearby neighborhoods or distant to improve the current solution. Thus, if the search is carried out in a close neighborhood, the method is called local search and is noted, LS (Local Search) and in the case where research is carried out in a wider neighborhood, the method is called: the research in a broad neighborhood and is noted, LNS (Large Neighborhood Search). Nevertheless, it is also possible to mix all the neighborhoods research, in this case we are talking about research in a variable neighborhood VNS (Variable Neighborhood Search). For more details on the most widespread diversifying methods, (see Siarry [65]).
In a research process, we often find treatments that serve to diversify solutions. This diversification aims to explore other research sub-areas, which probably have not been visited. Among these diversification process, there are procedures based on disturbances of solutions. In our study, we were more interested in the disruption of solutions, which is to disturb the neighbourhood of current solutions [38].
In general, in order to improve the solution obtained by diversifying method, we start with the solution obtained by the greedy algorithm and if it is possible, choice set of components by the random method and apply these steps with several times, keeping each time the best solution obtained.

In QKP, binary solution are represented by item i (where $i \leq n$), a binary variable x_i is fixed to the value 1 (if it is involved in the solution) or to the value 0 (otherwise). Certainly, if its weight is less than or sufficient for the remaining capacity of the knapsack and also if it is compatible with the already fixed items, consequently, item i is contributed in the solution.

Diversifying approach seems to us more reliable, since we had well setting for the parameters related to the disturbance.
We denote with N_k $(k = 1, ..., k_{max})$, a finite set of pre-selected neighborhood structures, and with $N_k(x)$ the set of solutions in the k^{th} neighborhood of x. So, we select the set of neighborhood structures N_k,$k = 1, ..., k_{max}$, that will be used in the search.

Then, we start with the initial solution x and we choose a stopping condition. Let: $k \leftarrow 1$; l after, we repeat the following steps until the stopping condition is performed:

- Generate the solution $x^{'}$ from k^{th} neighborhood of x $(x^{'} \in N_k(x))$.
- Achieve the local search approach by $x^{'}$ as initial solution to obtain $x^{''}$ as local optimum.
- If this local optimum is better than the incumbent, move there $(x \leftarrow x^{''})$, and continue the search with $N_1(k \leftarrow 1)$, otherwise set $(k \leftarrow k + 1)$.
- Until $(k = k_{max})$.

Metaheuristic methods

The term metaheuristic was first introduced in Glover [23]. Metaheuristic is a higher-level procedure or advanced heuristics for many combinatorial optimization problems. So, it encourages and guides the search process to explore new regions in the solution space beyond local optimum. Then, it produces high-quality solutions and not necessarily optimal. In general, metaheuristics are based on two mechanisms:

- intensification, which is the mechanism by which they find the local optimum of the current solution (intensification often amounts to local search).
- diversification, which is the mechanism allowing the metaheuristics to leave the neighborhood of this local minimum (diversification varies from one algorithm to another).

The best metaheuristics methods gives an excellent results on large-scale problems with reasonable run times which is contrary to the exact methods.

We noticed two large families of metaheuristics: metaheuristics that manipulate one solution at a time, called metaheuristic "single solution" and metaheuristics that manipulate a multitude of solutions, called metaheuristics to "population of solutions" (multi-solutions).

There are several mtaheuristic methods with single solution that well-known in literature (see Siarry [65]), such as: the procedure of greedy search randomise, note: GRASP (Greedy Randomized Adaptive Search Procedure), itrative local search, note: ILS (Iterated Local Search), variable neighborhood search, note:

VNS (Variable Neighborhood Search), simulated annealing, not: SA (Simulated annealing) and the tabu search, note: TS (Tabu Search)(see [25]).
We summarize the principles of these single solution metaheuristics in the following lines:
GRASP which is introduced in Feo *et al.* [6], creates a new solution to each iteration using a random greedy algorithm and improves this solution with a local search. Metaheuristics (GRASP) stops after a predefined number of iterations and the best solution found during the different iterations is saved as a result.

With regard to iterative local search (ILS), the principle is similar to that of GRASP, except that the initial solution of each iteration is obtained by applying a random turbation to the best solution found so far. The idea is to keep some features of the best solution [40].

Variable Neighborhood Search (VNS) presented in Mladenovic and Hansen [69], is also inspired by GRASP, but uses a series of k neighborhoods. The draw of the new solution and the local search are carried out alternatively on the different neighborhoods. Knowing that several variables of a local optimal solution keep the same value in a global optimal solution, so, exploring different neighborhoods set of variables at once, in order to find the global optimum.
Simulated annealing (SA) is inspired by the thermal annealing process used in metallurgy in order to reduce the energy of a material by alternating phases slow cooling and reheating phases (annealing). It was introduced in combinatorial optimization as metaheuristic in [79].
The tabu search (TS), is a metaheuristic that consists in traversing all the neighborhood of the current solution, then apply to it bans on the return, even if the latter degrades the objective function. This metaheuristic was introduced in Glover and Laguna [25].
The representative applications of TS or tabu mechanism to quadratic knapsack problems include GRASP+tabu algorithm for the QKP, and the tabu enhanced iterated greedy algorithm (see Yuning *et al.* [37]).

The most popular solutions to population problems are: search scattered, note: SS (Scatter Search), swarm optimization of particles, note: PSO (Particle Swarm Optimization) [83], optimization by ant colonies, notes: ACO (Ant Colony Optimization) and the genetic algorithm, not: GA (Genetic Algorithm). For more details on these metaheuristics, (see Siarry [65], [11]).
In follows, we will recall the principles of metaheuristics to populations of solutions:
Scattered Search (SS) is a population-based method that uses local search and other operators allowing a combination of solutions (see Marì *et al.* [26]). Thus, at each iteration, a subset of solutions of which the elements meet quality and diversity criteria is selected and, is called reference set. The solutions of the different reference sets are then combined to give new solutions, which are then improved by local search.
Particle swarm optimization (PSO) algorithms are inspired by insects swarms and their coordinated movements. Of the same way that these birds move in

groups to find food or to avoid predators, particle swarm algorithms are looking for solutions for an optimization problem. The individuals in the algorithm represent solutions and are called particles, while the entire population is called a swarm [8].
The genetic algorithm (GA), presented in Holland [31], is inspired by the genetics and natural evolution of species that use crosses, selections and mutations to move towards optimum, a population of solutions represented as chromosomes. The genetic algorithm stops after a predefined number of iterations. At each iteration, two parent chromosomes of the population are selected favouring the most promising, then crossed to obtain one or two child chromosomes combining the characteristics of the parents. However, mutations can be applied to children as diversification to avoid premature convergence of the population. (for more details on GA, the reader can refer to the book [31]).
In fact, there are several metaheuristics methods in existence, while a new variants are continually being proposed. So, some of them are developed and applied to a variety of combinatorial optimization problems such as: Osman *et al.* [62], Vaessens *et al.* [39], Hifi *et al.* [52], Blum *et al.* [12], Akeb *et al.* [32].

5.4 Conclusion

In this chapter we recover the literature review over the main resolution methods used to solve the combinatorial optimization problems.
In fact, we have underlined the remarkable evolution in the last few decades.
Thus, we have presented some of resolution methods used to solve combinatorial optimization problems in general, and the quadratic knapsack problems in particular.
The models and algorithms presented are stepping-stones on which improvements in the solution of combinatorial optimization problems can always be used to find better solutions.

In the next chapter, we successfully applied a combination of some diversifying methods to approximately solve the problem of QKP.

Chapter 6

Chapter 6

Reactive Search for the Quadratic knapsack Problem

Contents

In this chapter, we introduce a reactive method to solve the quadratic knapsack problem (noted QKP). The proposed method is mainly based on two complementary phases. In the first phase we use a greedy algorithm to provide the starting solution. In the second phase we improved the quality of the starting solution based on a reactive search with applying a destroy and repair process, tries to diversify and to enhance the solutions by using some parameters. The proposed algorithm is based both diversification and intensification strategy that computationally analyzed on a set of benchmark instances known in the literature. The results obtained are compared to those achieve by both the Cplex solver and a recent algorithm of the literature.

6.1 Introduction

In all the variants of the knapsack problems considered so far the profit of selecting a specific item was independent of the other selected items. One possible formulation of such an interdependence is the quadratic knapsack problem (QKP) in which an item has a corresponding profit and an additional profit is retrieved if the item is chosen together with another one. (see section 5.2 in chapter 5). In fact, the quadratic knapsack problem was first introduced by Gallo, Hammer and Simeone. It has been studied intensively in the last decade due to its simple structure and challenging difficulty and broadly applicability in both practical

and theoretical problems (see section 5.1 in chapter 5).
Therefore, to solve the quadratic knapsack problem, we investigate the use an adaptation of the reactive method. Our approach can be summarized in two complementary phases.

- The first phase serves to provide a starting solution using a greedy procedure that inserts an item which has best efficiency into the knapsack until filling it.
- The second phase is constructed by an intensification procedure in the neighborhood of the current solution. The neighborhood is obtained by removing some items from the current solution and inserting other ones whose overall weight does not exceed a given knapsack capacity (see section 5.3.2 in chapter 5).

The remainder of the chapter is organized as follows:
Section 6.2 explains the steps of the proposed reactive search. Subsection 6.2.1 presents the principle of the greedy algorithm that we used to construct a starting solution. Subsection 6.2.2 presents the steps of the reactive algorithm that we used to improve the starting solution. Section 6.3 analyses the performance and results of the proposed approach. Section 6.4 gives a description of instances used in this thesis including those commonly used in the literature and some new large instances proposed. And finally, section 6.5 concludes by summarizing the main contribution of this method.

6.2 Reactive search for the quadratic knapsack problem

The main objective of this work is to develop effective methods to solve large QKP cases in a limited time. For this end, we implemented an adaptation of reactive method. The adaptation we have implemented is mixed and it is based on neighborhood search to improve the starting solution of the problem. A neighborhood search that embeds a new local search is performed through diversification of an initial solution obtained by an greedy search. This approach uses two complementary procedures:

- **(1)** Greedy algorithm to construct the starting solution. This procedure builds a solution by selecting an efficient item i from set of candidates and introduces it into the knapsack until filling it and sets it to the starting solutions.
- **(2)** Reactive search tries to improve the quality of the starting solutions. This procedure uses a perturbation method that consist of two strategies:

1. **Degrading strategy:** Destroy part of the current solution in order to provide a partial solution.

2. **Reconstruction strategy:** Complete the current partial solution in order to provide a complementary solution and re-optimize it by linear optimization program to provide a new diversified solution.
   Concerning the diversification of the solutions, we applied a process based on the neighborhood search. This method discovers a set of solutions that are in the neighborhood of current solution during solving. For more details about diversification strategy (see algorithm 11).

6.2.1 Greedy algorithm to construct a starting solution

There are several ways to define a starting solutions of the QKP, but we used an intuitive approach considering the profit to weight ratio e_i of each item, also called the efficiency of an item (see [78]). We can formulated as the follows:

$$e_i = \frac{p_i}{w_i}$$

and we try to include items with highest efficiency into the knapsack. Clearly, these items generate the highest profit while consuming the lowest amount of capacity. Therefore, all items are sorted in descending order according to their efficiency as :

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_n}{w_n}$$

Algorithm 10 summarizes the construction of a starting solution.

Algorithm 10 Greedy method

Input: Instance of QKP.
Output: Starting solution (S).

Let c the capacity of the knapsack;
Sort $(e_i = \frac{p_i}{w_i})$ in *decreasing* order;
Let $Z \leftarrow 0$;
for (each item $i \leq N$) **do**
  for (each item $j \neq i$) **do**
    if $(w_i \leq c)$ **then**
      $x_i = 1$;
      $c = c - w_i$;
      $Z = Z + P_{ij} + P_{ji}$;
    else
      $x_i = 0$;
    end if
  end for
end for

As can be seen in our algorithm above, we focus on the descending order of efficiency items to fill the knapsack. The idea of the greedy algorithm with solution value is to start with an empty knapsack and simply through the items in this descending order of efficiencies. In each step, an item under consideration is selected, then it is included in the knapsack if the capacity constraint is not violated thereby. Otherwise, we select the next item that may be acceptable. This process is repeated until filling of the knapsack (see section 5.2 in chapter 5).

A starting solution must be generated in order to even start a reactive search. Greedy methods are a class of heuristic (or approximate) solution methods that construct a solution by immediate improvement without considering the consequences. In other words, they make always, a local optimum choice in the hope of reaching a global optimum solution.

It is important that the method used is usually extremely fast, yield the same solution at every execution on the same instance and it is the most immediate way to determine a feasible solution.

6.2.2 Neighborhood search to improve a starting solution

In this work, we can improve the starting solution by using the neighborhood algorithm that combines two strategies: *degrading* and $re-constructing$ strategies (as used in Hifi and Michrafy [52]), but we have improved this process by making a re-optimization of the overall solution and by integrating a list of elements not to be included in the selection.

In the following, we will show how to combine both strategies for searching a series of new solutions. Let S be the starting solution obtained from algorithm 10. Let α be a constant, such that $\alpha \in [0, ..., 100]$, denoting the percentage of packed items of the set N, i.e., some items are loaded to knapsack according to the current solution S. Then, the diversification procedure can be considered as an alternate approach which is performed by using two procedures that are:

- The **degrading** strategy is used to fixed $\alpha\%$ items from the current solution for extract a partial solution $S^{'}$.
- The $\mathbf{re-constructing}$ strategy is used to complete the partial solution obtained after destruction and re-optimize the complementary solution $S^{''}$ by application the Cplex solver on the subset $S^{''}$ and to improve the current solution. This procedure is based on destroy of a some items from the current solution and replaced with the items of its neighbours whose overall weight does not exceed a given knapsack capacity.

The details of this procedure presented in algorithm 11 that started by consider the starting solution obtained by algorithm 10 as a best global solution. The first loop in line 2 is used to stop the search when the stopping criteria is performed; herein, a maximum number of iterations is considered. The best solution is setting to S_{local} in line 3 as global solution we want to optimize by

Algorithm 11 Neighbourhood algorithm

Input: Starting solution S of greedy algorithm 10.
Output: An approximate solution S_{global}.

1: $S_{global} \leftarrow$S
2: **while** (Stopping criterion is not performed) **do**
3: $S_{local} \leftarrow S_{global}$
4: **while** $(I < Max_{Iteration})$ **do**
5: Call degrading procedure in order to provide a partial solution $S^{'}$
6: Call re-optimize procedure in order to provide the complementary solution $S^{''}$ and for realizing a new solution S_{local}
7: **if** $(S_{local} > S_{global})$ **then**
8: $S_{global} \leftarrow S_{local}$
9: **end if**
10: **end while**
11: **end while**

Algorithm 12 Degrading Procedure

Input : S_{local} local solution of algorithm 11.
Output : $S^{'}$ partial solution of S_{local} .

1: $\alpha \leftarrow [0, ..., 100]$.
2: Consider X_j as the packed items set associated to the current solution S_{local}.
3: Select $\alpha\% \in X_j$ belonging to the current solution S_{local}.
4: Fixed $\alpha\%$ of variables X_j to value 1.
5: Extract the partial solution $S^{'}$.

Algorithm 13 Re-optimize Procedure

Input : $S^{'}$ partial solution of algorithm 12.
Output: S_{local} new local solution .

1: Find the complementary solution $S^{''}$.
2: Call Cplex solver to optimize the complementary solution.$S^{''}$.
3: Combine both *partial solution* $S^{'}$ and *complementary solution* $S^{''}$ for realizing a new solution S_{local}

combining both degrading and re-optimizing strategies. The second loop in line 4 serves as local loop for search the best local solution on a series of neighborhoods. Next, the degrading procedure is called in line 5 in a local loop for fixing $\alpha\%$ of variables that have value 1 from the current solution and replace the others items with new ones that are not included in the knapsack, trying to improve the quality of solution by the re-optimization procedure in line 6. Then, the best global solution is compared with the best local solution to extract the best feasible solution. Similarly, this approach can be repeated until reaching a maximum number of diversification or limiting overall execution time for the judgement of the method.

6.3 Computational experiments

In this section, the most significant results are presented. Our goal is to show and discuss the results of the above algorithms in section 6.2 that effectively allows to approximately solving the quadratic knapsack problem(QKP). The computational results are divided into two parts:

- The first part presented in table 6.2, shows solutions of greedy algorithm presented in algorithm 10.
- The results of second part presents the results for the reactive algorithm presented in algorithm 11. This part studies effect of destroy process on the quality of the final solution produced by neighborhood search algorithm(see table 6.3).

6.4 Benchmark instances

The benchmark instances used in this thesis include those commonly used in the literature and some new large instances proposed. The instances that we used to test our proposed algorithm for the QKP can be divided into three sets:

- **QKPSet I.** This set is composed of 100 small and medium sized benchmark instances generated randomly by Alain Billionnet and Eric Soutif [2]. These instances are very popular and used to test many QKP algorithms. These instances are characterized by their number of objects $n \in \{100, 200, 300\}$, density $d \in \{25\%, 50\%, 75\%, 100\%\}$. Each (n, d) combination involves 10 different instances distinguished by their labels except for $(300, 75\%)$ and $(300, 100\%)$ where instances are missing. Optimal solutions are known for these instances.

  Table 6.1 describes the main characteristics of these set of instances. Column 1 of the table indicates the label of instance. Column 2 indicates the number of items for each instance, column 3 indicates the density (d) that associated to each class of the instance .

- **QKPSet II.** The second set includes 80 large-sized benchmark instances which are recently generated by Yuning *et al.* [34]. These instances have a number of objects from 1000 to 2000, a value of density from 25% to 100%. Due to their large size, optimal solutions are still unknown for these instances.

- **QKPSet III.** This set is composed of 40 new instances of very large sizes that are proposed by Yang Chen [34]. They are characterized by their number of objects $n \in \{5000, 6000\}$ and density $d \in \{25\%, 50\%, 75\%, 100\%\}$. For each (n, d) combination, 5 instances were proposed.

Inst.	*N*	*Density*
I1.1	100	25
I1.2	100	50
I1.3	100	75
I1.4	100	100
I2.1	200	25
I2.2	200	50
I2.3	200	75
I2.4	200	100
I3.1	300	25
I3.2	300	50
I4.1	1000	25
I4.2	1000	50
I4.3	1000	75
I4.4	1000	100
I5.1	2000	25
I5.2	2000	50
I5.3	2000	75
I5.4	2000	100
I6.1	5000	25
I6.2	5000	50
I6.3	5000	75
I6.4	5000	100
I7.1	6000	25
I7.2	6000	50
I7.3	6000	75
I7.4	6000	100

Table 6.1: The description of instances for the QKP

6.4.1 Parameter settings

We note that Cplex 12.6, [41] was run on a multi-users ubuntu laptop (2.5 GHz, 5 Gb), with 3600 seconds cpu time limit. The proposed algorithms was coded in C++ and run on the same machine with cpu time limit of, 20 seconds for all instances. In this chapter, for both (Cplex and proposed algorithms), we use the instances of Billionnet *et al.* [2].

6.4.2 Test the performance of proposed methods

In this section, we investigate the behaviour of our method, the first experiment is performed on the benchmark instances of **QKPSet I.** (see section 6.4). Table 6.2 summarizes the results obtained in initialization mode using greedy algorithm on 100 benchmark instances with single iteration.

Column 1 displays the label of each class. The averages values of the starting solutions are presented in column 2. Column 3 displays the averages run time for each class.
Note that:

- The instance $(100 - 100 - 4)$ is not available and is not considered when we calculate the average value for the instance class $(100 - 100)$.
- The instance $(300 - 25 - 3)$ is not available and is not considered when we calculate the average value for the instance class $(300 - 25)$.

The analysis of table 6.2 as:

1. We recall that our problem is a maximization problem. In this context, quality of solutions produced by a greedy search actually a good solutions.
2. We note that the average runtime of a greedy search is very small for all instances.

Inst.	*Avg.Sol.*	*Avg.t(s)*
I1.1	22157.8	0.003
I1.2	45863.7	0.001
I1.3	74274.5	0.001
I1.4	133138.56	0.002
I2.1	88289	0.003
I2.2	193709	0.003
I2.3	183731.6	0.004
I2.4	413627.2	0.003
I3.1	157334.67	0.007
I3.2	410492.3	0.006
Avg.	172261.8	**0.003**

Table 6.2: Performance of the greedy algorithm on 100 benchmark instances with single iteration

6.4.3 Effect of the destroy and repair process

In this section, reactive search results are presented that provided by algorithm 11. Herein, it is able to interpret the effect of remove some items from the current solution on the quality and speedy of the final solution. However, the reduction process greatly accelerates the execution of our algorithm especially for **QKPSet I.** This set is composed of 100 small and medium sized benchmark instances. Table 6.3 summarizes the results obtained when we vary the destroy percentage. Column 3 shows the quality of the final solution when the random destruction percentage is $\alpha = 5\%$,

column 4 shows the quality of the final solution when the random destruction percentage is $\alpha = 10\%$, and finally, column 5 shows the quality of the final solution when the random destruction percentage is $\alpha = 20\%$. The analysis of table 6.3 shows that destroy of $\alpha = 20\%$ of the previous solution seems the most effective setting.
Table 6.4 summarizes a comparison of the average cpu time of the computational results obtained for each percentage. We note, also, that the percentage of $\alpha = 20\%$, provided the better solution and less runtime compared with the solutions and runtime generated in the case percentage of $\alpha = 5\%$ and $\alpha = 10\%$.

	Average solutions with variation of $\alpha\%$			
Inst.	*Best*	%5	%10	%20
I1.1	32254.8	32254.8	32254.8	32254.8
I1.2	66571.85	66571.85	66549.75	66511.05
I1.3	96992.55	96992.55	96992.55	96992.55
I1.4	144569.78	144545.56	144559.78	144569.78
I2.1	126890.65	126651.15	126762.85	126890.65
I2.2	287388.6	287359.9	287374.8	287388.6
I2.3	281742.1	281324.4	281561.5	281742.1
I2.4	468591.9	467646.5	468351.1	468591.9
I3.1	241039.11	240819	240974.89	241039.11
I3.2	583125.2	581905.3	582838	583125.2
Avg.	232916.65	232607.10	232822.00	**232910.57**

Table 6.3: Performance of reactive search results with random selection policy and 20 seconds cpu time limit

	Average time with variation of $\alpha\%$			
Inst.	*Best*	$t(s)_{\%5}$	$t(s)_{\%10}$	$t(s)_{\%20}$
I1.1	1.65	2.35	2.06	1.65
I1.2	10.16	10.16	10.21	7.96
I1.3	23.34	66.50	44.87	23.34
I1.4	35.30	63.70	50.90	35.30
I2.1	36.86	42.99	41.56	36.86
I2.2	108.94	129.93	110.60	108.94
I2.3	176.60	135.20	164.20	176.60
I2.4	169.91	180.37	122.37	169.91
I3.1	58.14	80.70	89.33	58.14
I3.2	237.11	236.08	232.18	237.11
Avg.	85.80	94.80	86.83	**85.58**

Table 6.4: Average cpu time of the computational results for reactive search

Table 6.5 shows some results of the best method in the literature (Alain Billionnet and Eric Soutif [2]). Noted that, the solution provide by our approach is approximated to the optimal solution realized by Cplex solver and the best solutions given by the literature, with less run time. We can observe the Cplex solver is able to match the best solutions of the literature, but with more runtime especially with the large size instances. In the other hand, our approach can lead to an efficient and fast solution when the solution given by the first phase is good.

Inst.	$Sol._{RS}$	Cpu_{RS}	Opt_{cplex}	Opt_{Lit}	$Cpu_{Lit.}$
I1.1	32254.8	1.65	32254.8	32254.8	108.18
I2.4	468591.9	169.91	468919	468914.80	70633.00
I3.1	241039.11	58.14	241044.78	241044.78	2074,09
I3.2	583125.2	237.11	583270	583270	2889.927
Avg.	331252.75	**116.70**	**331372.15**	331371.10	24543.70

Table 6.5: Comparison of some numerical results obtained by (RS) versus Cplex solver and literature

6.5 Conclusion

In this chapter, we are introduced a reactive search to solve the quadratic knapsack problem that is mainly based on two complementary phases such as:

- A greedy strategy that include an item with highest efficiency into the knapsack to construct a starting solutions.
- A Neighborhood search to improve the quality of the starting solutions.

We use a hybrid method by combining two phases, both serves to intensify and diversify the search process. The proposed approach begins by constructing an initial solution using a greedy strategy with one iteration. After, we used the neighborhood search to improve the quality of solutions and produce the best solutions. So, the proposed approach implements a diversification of solutions with fixed iteration number. The numerical results showed that the proposed approach provides encouraging results by improving the quality of solutions for known instances of the literature (Alain Billionnet and Eric Soutif [2]) that compared with the solutions provided by Cplex.12.6 solver. The performance of the proposed approach was evaluated based on the set of the benchmark instances of the literature.

As shown from the experimental results, according to our preliminary computational results, the behaviour of the cooperation is encouraging. The

obtained results were compared to those reached by known literature, this results are efficient and fast when the solution given by the greedy procedure is good. Further, from the experimental results, the quality of solutions based on the well setting of the parameters such as (α, cpu time and iterations number) that are experimentally based the characteristics of the instances.

The reactive search is published in an international conference with proceedings 4th IEEE Control, Decision and Information Technologies, (CoDIT 2017) [57].
From the work presented in this chapter it is necessary to draw attention to the primary goal that was considered when this research started. The primary goal was to develop solution approaches based upon neighborhood search techniques tailored for optimizing large size instances of combinatorial optimization problems. The general outcome, according to the experimental analysis on benchmarks instances of the literature, is that neighborhood search solution techniques are successful in developing better algorithms to approximate large and very large size instances of the considered problem in this work.
In order to develop our algorithm we propose another direction of search to solve quadratic knapsack problems (especially for large and very large size instances (see section 6.4) through variables reduction search based upon fix α and β variables from the binary solutions of reactive search, section 6.2. we will present this work in the following chapter.

Chapter 7

Chapter 7

Solution of Large-sized Quadratic Knapsack Problems Through Variables Reduction Search

Contents

From the work presented in chapter 6 and according to the experimental analysis on all benchmarks instances of the literature, it is necessary to introduce some development on our algorithm to provide an efficient resolution method yielding high quality solutions with fast runtime to approximate large-size and very large instances of the quadratic knapsack problem (QKP).

In this chapter we propose a hybrid algorithm to solve the quadratic knapsack problem (QKP). The aim of this algorithm is to improve our algorithm that is able to guide the search process towards new search spaces. The method combines the principle of a neighborhood search and a strategy of variables reduction search. First, the strategy of variable reduction is used to construct a series of sub-problems to solve. Then, the method uses an optimization by ILP, Cplex solver.

Consequently, the experimental results show the effectiveness of the proposed approach to provide optimal solutions for 100 small and medium sized benchmark instances that were investigated in chapter 6. As shown

from the computational results, the proposed algorithm is capable of solving the small, large and very large size instances of the QKP that cannot be solved by Cplex solver and by other methods of the literature.

7.1 Introduction

The quadratic knapsack problem (QKP) is the generalization of the classical 0-1 knapsack problem obtained when the objective function is permitted to be quadratic (see chapter 5). Several reduction methods used for fixing some of the decision variables at their optimal value, have been presented for the QKP (see Caprara, Pisinger and Toth [5])and (P.L. Hammer and D.J. Rader Jr [70]). Recent advances in the solution of difficult NP-hard optimization problems have shown the importance of good reduction techniques. These 0-1 QKP can be solved by a specialized dynamic programming or branch-and-bound algorithm (see Billionnet, Faye and Soutif [1]), but in practice, one can often solve them easily and quickly using a general-purpose ILP solver.
Following this direction of research; in this chapter, we present an approximate approach to solve the QKP, where reduction plays a key role in the solution process. However, it is much fast and simpler, since it does not require more complex procedures. The proposed approach is based on the same algorithms as those presented in chapter 6 of this thesis, and we applied some modifications that will be presented in the following sections.
The remainder of this chapter is organized as follows: section 7.2 reviews the principles of the large neighborhood search. Section 7.3 describes the principle of the proposed approach. In section 7.4, the performance of the proposed approach is evaluated on a number of instances of the literature, and analyzes the obtained results. Finally, section 7.5 summarizes the contents of the chapter.

7.2 Large neighborhood search(LNS)

Solving combinatorial optimization problems using exact methods becomes discouraged when the structure of problems is complex. Indeed, proving optimality of problems requires generally a huge computational resource (Papadimitriou and Steiglitz [30]). Contrast to the fact that the exact methods aim at solving problems by proving their optimality, the approximative methods focus only on the computation of high quality solutions (near optimal) with reasonable computational effort. Large Neighborhood Search (LNS), proposed by Shaw [67], has been considered as one of the most efficient algorithms for solving large-scale optimization problems (Ahuja and Ergun [58]).

Unlike the approximate methods that explore generally the whole solution space, LNS consists of finding high quality solutions by randomly exploring a series of sub-solution spaces, where each subspace is characterized by a neighborhood of a local optimum [18].
Generic schema of LNS consists of two strategies: *reduction strategy* and *exploring strategy*. More precisely, the reduction strategy is used to yield a sub-solution space (i.e., the neighborhood of the local optimum) while, the exploring strategy is then applied in order to improve the current local optimum in its neighborhood [53].

7.3 Variables reduction search(VRS)

The binary quadratic knapsack problem (QKP) was introduced by Gallo *et al.* [28], formally as we defined it in chapter 5. The decision variables x_i at their optimal value, we have presented for the QKP in chapter 6, that have either the values 1 ($x_i = 1$), when choosing the item to include in the knapsack or the values 0, ($x_i = 0$), when choosing the item to exclude in the knapsack. The binary feasible solution can be represented by series of the binary variables x_i takes the value 1 if and only if the ith item is inserted into the knapsack.
However, the solution space of the problem QKP might be too large, to be efficiently searched by an algorithm, the size of a QKP problem may be considerably reduced by using some reduction rules such that, one may obtain a feasible solution by truncating the some variables from the search space [73].
To further reduce the search space to explore, we fix some variables to a manageable size. The reduction, including an improved version of our algorithm we have presented for the QKP in chapter 6, that fix $\alpha\%$ from the binary variables x_i that takes the value 1, to further reduce the search space to explore, we fix $\beta\%$ from the binary variables x_i that takes the value 0. Consequently, the remaining set of variables are easily handled through the Cplex solver [41]. Algorithm 14 summarizes the main steps of a generic schema of a hybrid algorithm (noted as HLNS).

The main loop (steps 2-20) of HLNS applies alternatively the reduction strategy (steps 5-13) and the exploring strategy (step 14) in order to improve the current local optimum. The reduction procedure (steps 5-13) serves to fix randomly a limited number of items, its decision variables values have been determined, in order to yield a reduce solution of QKP and a partial solution (i.e., with some decision variables have been set free). The exploring procedure (step 14) aims to improve the current local optimum by exploring the solution space yielded from (steps 5-13). HLNS (algorithm 14) exits with the best solution found so far when the iteration limit is reached.

Algorithm 14 An hybrid algorithm to solve the quadratic knapsack problem (QKP)

Input: A starting feasible solution S 0-1 QKP
Output: A best global solution S_{global}

$S_{global} \leftarrow$ S
while (the iteration limit is not performed) **do**
  $S_{local} \leftarrow S_{global}$
  while ($I < Max_{Iteration}$) **do**
    $\alpha \leftarrow [0, ..., 100]$
    $\beta \leftarrow [0, ..., 100]$
    Consider X_i as the packed items set associated to the current solution S_{local}
    select $\alpha\% \in X_i$ belonging to the current solution S_{local},where $X_i = 1$;
    select $\beta\% \in X_i$ belonging to the current solution S_{local},where $X_i = 0$;
    Fixed $\alpha\%$ of variables X_i to value 1
    Fixed $\beta\%$ of variables X_i to value 0
    if ($\alpha\% > 0$) and ($\beta\% > 0$) **then**
      Extract the partial solution $S^{'}$
      Apply Cplex solver on the reduced binary solution in order to complete the partial solution $S^{''}$ and for realizing a new solution S_{local}
      if ($S_{local} > S_{global}$) **then**
        $S_{global} \leftarrow S_{local}$
      end if
    end if
  end while
end while

7.4 Computational results

This section evaluates the effectiveness of the proposed *Hybrid Large Neighborhood Search-Based Variables Reduction* (HLNS) on three sets of benchmark instances those presented in chapter 6 of this thesis.
The first group contains 100 small and medium sized benchmark instances generated randomly by Alain Billionnet and Eric Soutif [2], with $n \in \{100, 200, 300\}$ items, density $d \in \{25\%, 50\%, 75\%, 100\%\}$ (see table 7.1) and the second group contains 80 large-sized benchmark instances which are recently generated by Yuning *et al.* [34], where each instance contains 1000 to 2000 items, a value of density from 25% to 100% and, the third group contains 40 new instances of very large sizes that are proposed by Yuning Chen [34], where each instance contains $n \in \{5000, 6000\}$ items and density $d \in \{25\%, 50\%, 75\%, 100\%\}$.

Table 7.1 summarizes the average solution values realized by HLNS when compared to the average solution values obtained by reactive search within 20 seconds cpu time limit. Table 7.1 showed the average objective values

	CPLEX	RS		HLNS		BEST	
	Sol	Sol	time	Sol	time	Sol	time
Inst.	15000s	20s		20s		20s	
I1.1	32254.8	32254.8	1.65	32254.8	0.87	32254.8	0.87
I1.2	66571.85	66571.85	10.16	66571.85	1.41	66571.85	1.41
I1.3	96992.55	96992.55	23.34	96992.55	2.01	96992.55	2.01
I1.4	144570.22	144569.78	35.30	144570.22	1.31	144570.22	1.31
I2.1	126963.65	126890.65	36.86	126963.65	4.23	126963.65	4.23
I2.2	287390.8	287388.6	108.94	287390.8	7.24	287390.8	7.24
I2.3	281742.7	281742.1	176.60	281742.7	14.16	281742.7	14.16
I2.4	468918.9	468591.9	169.91	468918.9	23.82	468918.9	23.82
I3.1	241044.78	241039.11	58.14	241044.78	7.01	241044.78	7.01
I3.2	583270	583125.2	237.11	583270	23.10	583270	23.10
Av.	232972.025	232916.65	85.80	232972.025	8.516	232972.025	8.516

Table 7.1: Comparison results of both RS and HLNS with the Cplex solver for 20 seconds cpu time limit

(i.e., on each class of the benchmark instances) related to the best solutions when applying the Cplex solver (version 12.6), reactive search (RS) and HLNS respectively. Column 1 displays the name of each class of the benchmark instances. The average solution values provided by Cplex solver within 15000 seconds are shown in column 2. As VRS applies a random strategy to fix items, 20 independent trials of HLNS were performed for each instance with a runtime limit fixed to 20 seconds. According to the numerical results, the number of items to be fixed is experimentally limited to $\alpha\%$ and $\beta\%$ respectively, where $\alpha\%$ denotes the number of variables belonging to the set of binary decision variables x_i that will be fixed to value 1 and $\beta\%$ denotes the number of variables belonging to the set of binary decision variables x_i that will be fixed to value 0. Column 3 and column 4 tally respectively the average solution values and the average runtime of the reactive search algorithm with the 20 trials. In order to evaluate the performance of HLNS, the CPU runtime is limited to 20 seconds (i.e., the same value used by RS) whereas the maximum number of fixed items performed by HLNS on each instance is randomly generated in discrete interval $[0, ..., 100]$. Column 5 and column 6 show respectively the average solution values and the real run times realized by HLNS.

We also reported in column 7 and column 8 respectively the average of best solution and the average of best runtime over 20 trials for our HLNS algorithm.

As shown in table 7.1, the results provided by HLNS outperform generally those provided by the RS. On the other hand, HLNS matches the average solution values reached by Cplex solver. However, HLNS is much faster.

From table 7.1, we observed that our HLNS algorithm attains the known optimal values with a successful rate of 100% for all these instances with an average computing time 8.516 seconds. Another interesting feature of HLNS is that its average computing time is approximately linear relative to the size of the instance.

7.4.1 Comparative results on small and medium instances of Group I

The experiment is performed on the benchmark instances of Group I. These instances were first solved to optimality by the exact algorithm of [2]. Several recent heuristic approaches are able to attain these optimal solutions. To evaluate the performance of our HLNS algorithm, three leading heuristic methods were considered for our comparison: DP+FE algorithm [21], GRASP+tabu [20] and IHEA [36]. Table 7.2 shows our HLNS when compared with 3 state-of-the-art algorithms on the 100 benchmark instances of Group I. The results of the reference algorithms are extracted from the corresponding papers [20, 21, 36]. Where the relative percentage deviation (RPD), the average gap between the best lower bound and the best solution value in percentage over 100 trials.[1]

	DP+FE [21]		GRASP+tabu [20]		IHEA [36]		HLNS		Opt
Inst.	$t(s)$	RPD	$t(s)$	RPD	$t(s)$	RPD	$t(s)$	RPD	$Avg(best)$
I1.1	0.697	0.319	0.060	0.000	0.325	0.000	0.87	0.000	32254.8
I1.2	0.708	0.018	0.057	0.000	0.253	0.000	1.41	0.000	66571.85
I1.3	0.704	0.008	0.052	0.000	0.334	0.000	2.01	0.000	96992.55
I1.4	0.657	0.005	0.048	0.000	0.248	0.000	1.31	0.000	144570.22
I2.1	7.341	0.056	0.286	0.000	0.714	0.000	4.23	0.00	126963.65
I2.2	8.239	0.009	0.301	0.4E-6	0.827	0.000	7.24	0.000	287390.8
I2.3	7.055	0.010	0.318	0.000	0.946	0.000	14.16	0.000	281742.7
I2.4	6.683	0.004	0.251	0.000	0.722	0.000	23.82	0.000	468918.9
I3.1	28.341	0.061	0.735	0.001	1.122	0.000	7.009	0.000	241044.78
I3.2	31.324	0.003	0.763	0.000	1.156	0.000	23.10	0.000	583270
Av.	9.18		0.29		0.67		8.52		232972.025

Table 7.2: Comparative results of HLNS on small and medium benchmark instances of Group I

From table 7.2, we can observe that: HLNS dominates the deterministic DP+FE algorithm in terms of both quality of solution and computational efficiency. Compared to GRASP+tabu which is one of the current best performing heuristic algorithms, IHEA remains very competitive since it solves all these instances to optimality.

[1]The gap is calculated by $((f_{LB} - f_{best})/f_{LB} \times 100)$.

7.4.2 Comparative results of HLNS on large sized instances of Group II

In this section, in order to highlight the effectiveness of the proposed approach, the second experiment we are performed to investigate the behavior of our algorithm on the second group of 80 large instances (n = 1000 or 2000) items (**QKPSet II.** (see chapter 6)). Moreover, we decided to compare the results provided by HLNS to the best solutions of the literature (table 7.3). Table 7.3 shows performance of our HLNS when compared to

	DP+FE [21]		GRASP+tabu [20]		IHEA [36]		HLNS	
Inst.	*Avg.Best*	*Avg.t(s)*	*Avg.Best*	*Avg.t(s)*	*Avg.Best*	*Avg.t(s)*	*Avg.Best*	*Avg.t(s)*
I4.1	2016881.6	2441.353	2016960.7	21.868	2016960.7	5.489	2016960.7	562.813
I4.2	5118780.3	2862.216	5118953.9	31.173	5118953.9	6.022	**5118956.9**	1322.954
I4.3	5900488.2	3143.037	5900733	26.281	5900793.2	6.39	**5900796.2**	10615.923
I4.4	7476295.3	3224.189	7476264.3	32.533	7476457.7	6.117	**7476472**	2924.653
I5.1	7841070.5	50528.385	7841340.4	295.865	7841340.4	22.244	7841340.4	2957.049
I5.2	18616684.3	50427.949	18617353.5	352.216	18617410.4	23.240	**18617420.1**	7621.856
I5.3	24676067.1	52334.3	24675823	287.102	24676371.6	22.718	24676371.6	28670.069
I5.4	21220000	53492.382	21218680.5	383.415	21220476.4	22.719	**21220476.5**	10799.255
Avg.	11608283.413	27306.727	11608263.663	178.807	11608595.538	14.367	**11608599.3**	8184.321

Table 7.3: Performance of HLNS with 30 seconds cpu time limit on the large-sized instances of Group II.

3 state-of-the-art algorithms on the 80 benchmark instances of Group II. Table 7.3 summarizes the results based on average best solution value ($Avg.Best$) and average computing time ($Avg.t(s)$) respectively. Column 1 displays the name of each class of the instances. The last row of the table, ($Avg.$) indicates the average of the listed values of each column.
From table 7.3, we can observe that:

(a) As shown from the experimental results, the HLNS was able to provide high quality solutions for all large size instances.

(b) We observed in this table, the results provided by HLNS outperforming generally those provided by the literature DP+FE [21] for all instances.

(c) The average solution value realized by HLNS is better than the average solution value of both GRASP+tabu [20] and IHEA [36].

7.4.3 Comparative results of HLNS on 40 very large instances of Group III

Table 7.4 shows performance of our HLNS when compared to the best algorithms of the literature IHEA [36] on the 40 very large-sized instances of Group III.

- As shown from the first experimental results, the HLNS was able to provide high quality solutions for all class of the instances.
- On the other hand, the high quality solutions provided by HLNS to solve very large-sized instances can be realized by extending the cpu run time limit to 300 second.

	IHEA [36]		HLNS	
Inst.	*Avg.Best*	*Avg.t(s)*	*Avg.Best*	*Avg.t(s)*
I6.1	34655399.6	130.41	**34655402.8**	57446.26
I6.2	91637046.6	128.05	**91637183.2**	54756.05
I6.3	116348325.2	135.5	**116348331.2**	43717.52
I6.4	87540027.8	109.40	87540027.8	58568.91
I7.1	76096536.8	204.39	**76096544**	52270.39
I7.2	163300152	218.40	**163300154.4**	69258.95
I7.3	208559651.2	226.30	208559651.2	40881.87
I7.4	386197890.2	240.13	386197890.2	51329.59
Avg.	145541878.675	174.075	**145541898.1**	53528.69

Table 7.4: Performance of HLNS with 300 seconds cpu time limit on 40 very large-sized instances of Group III.

7.5 Conclusion

In this chapter, we proposed a hybrid algorithm to solve the quadratic knapsack problem (QKP). This algorithm is able to guide the search process towards new search spaces. The hybrid method is based upon the principle of a large neighborhood search and a strategy of variables reduction search.
The proposed approach was introduced in order to develop our algorithm (RS) for providing an efficient resolution method yielding high quality solutions with fast runtime to approximate large-size and very large instances of the quadratic knapsack problem (QKP).
The performance of the proposed method was evaluated on the three sets of the standard benchmark instances of the literature. Then, the computational results showed that the proposed method is very competitive when compared to the Cplex solver and to the best recent method available in the literature. As shown from the experimental results, the proposed method was able to provide high quality solutions within fast runtime for the small, medium, and large instances, but with more runtime with the very large sized instances.

Chapter 8

Chapter 8

General conclusion

In this general conclusion, we present a brief summary and outline only the principal contributions of this work, since the detailed discussion of each contribution is presented as a final section of the corresponding chapter. In addition, we draw some perspectives on future work.

In this thesis, we studied two combinatorial optimization problems such: The problem of K-clusters in a bipartite graph, also known as "*K-Clustering minimum Biclique Completion Problem (K-CmBCP)*" and "*The quadratic knapsack problem(QKP)*".

At first, in order to draw some conclusions from the work presented in this thesis, it is necessary to draw attention to the primary goal of our work is essentially based on the propose approximate methods to effectively solve the two problems we are addressing.

Principles contributions of this work

In the first part of this thesis, our research concentrate on solving the problem of K-clusters in a bipartite graph. In chapter 3, after an overview the state of the art for the problem of K-clusters in a bipartite graph, we presented five mathematical models for the K-CmBCP, from which we obtained the linear mathematical model in (ILPM) which was then used in chapter 4 to obtain numerical results with the Cplex 12.6 solver based on the K-CmBCP instances that extracted from the literature.

In chapter 4, we proposed a hybrid method for approximately solving to the K-clustering minimum biclique completion problem. This approach uses a combination of three phases:

- In the first phase we proposed the sequential procedure that used for partition the services according to their clusters, and introduced the starting solution.
- In the second phase we proposed a neighbourhood search based heuristic as a local search that try to improve the quality of solutions.
- In the third phase, we used a perturbation procedure serve to intensify and diversify the search process.

The numerical results showed that the proposed approach provides encouraging results by improving the quality of solutions of the known literature and the quality of solutions provided by the Cplex.12.6 solver.

In the second part of this thesis, our research focused on the development of heuristic approaches to approximate large-size instances of NP-hard combinatorial optimization problems. Herein, we cited several algorithms for approximating a particular problem belonging to the knapsack family, the quadratic knapsack problem (QKP). In chapter 6, we introduced a hybrid reactive search to solve the quadratic knapsack problem that is mainly based on two complementary phases such as:

- A greedy procedure that iteratively introduces an item with best efficiency into the knapsack to construct a starting solutions.
- A neighbourhood search to improve the quality of the starting solutions.

The performance of the proposed method was evaluated on the set of the standard benchmark instances of the literature. From the experimental results, according to our preliminary computational results, the behaviour of the cooperation is encouraging the obtained results, when compared to those reached by Cplex solver and those solutions provided by known literature.
In order to develop our algorithm presented in (section 6.2 in chapter 6), in chapter 7, we proposed a hybrid algorithm to solve the quadratic knapsack problem (QKP). The proposed algorithm based upon the principle of a large neighborhood search and a strategy of variables reduction search (HLNS). This algorithm is able to guide the search process towards new search spaces.
The performance of the proposed algorithm was evaluated on the sets of the standard benchmark instances of the literature. The computational results showed the efficiency of the proposed algorithm when compared to the Cplex solver and to the best methods available in the literature.

The general outcome, according to the experimental analysis on benchmarks instances of the literature, is that hybrid search solution techniques are successful in developing better algorithms to approximate small and large-size instances of the considered problems in this work.

Perspective and future work

The work presented in this thesis would be adapted for the further development as well as, in order to improve our algorithm. We propose an approximate method to solve the problem of K-clusters in a bipartite graph: it is also interesting to consider other methods in order degradation and

reconstruction without using Cplex solver.
On the other hand, in order to improve our algorithm (HLNS) to solve quadratic knapsack problems, we consider another strategy to fix the best elements of binary solution to value 1 and to fix the worse elements of binary solution to value 0. Another suggested work is to consider other methods in order to develop our algorithm (HLNS) such as tabu search, genetic algorithm.

Bibliography

[1] Billionnet A., Faye A., and Soutif E. A new upper-bound and an exact algorithm for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 112:664–6727, 1999.

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

[3] Billionnet A. and Soutif E. Using a mixed integer programming tool for solving the 0-1 quadratic knapsack problem. *INFORMS Journal On Computing*, 16(2):188–197, 2004.

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

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

[6] Feo Thomas A, Resende Mauricio GC, and Smith Stuart H. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860–878, 1994.

[7] Meka Anand. *Distributed algorithms for summarizing and querying spatio-temporal data in sensor networks*. ProQuest, 2007.

[8] Eberhart Russ C and Kennedy James. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[9] Ferreira C., Martin A., De Souza C., and Wolsey L. Weismantel R. Formulations and valid inequalities for node capacitated graph partitioning. *Mathematical programming*, 74:247–266, 1996.

[10] Helmberg C., Rendl F., and Weismantel R. Quadratic knapsack relaxations using cutting planes and semidefinite programming. In *1996 Fifth IPCO Conference*, volume 1084, pages 175–189. Springer Verlag, 1996.

[11] Cotta Carlos, Mendes Alexandre, and Moscato Pablo. Memetic algorithms. In *New optimization techniques in engineering*, pages 53–85. Springer, 2004.

[12] Blum Christian, Roli Andrea, and Sampels Michael. *Hybrid metaheuristics: an emerging approach to optimization*, volume 114. Springer, 2008.

[13] Berge Claude. *Théorie générale des jeux à n personnes*, volume 138. Gauthier-Villars Paris, 1957.

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

[15] George Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[16] Pisinger David. *Algorithms for knapsack problems*. PhD thesis, University of Copenhagen, 1995.

[17] Pisinger David and Toth Paolo. Knapsack problems. In *Handbook of combinatorial optimization*, pages 299–428. Springer, 1999.

[18] Pisinger David and Ropke Stefan. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

[19] Alain F., Elloumi S., and Eric S. Decomposition and linearization for 0-1 quadratic programming. *Annals of Operations Research*, 99:79–93, 2000.

[20] Chu F., Wang G., and Yang Z. An effective grasp and tabu search for the 0-1 quadratic knapsack problem. *Computers Operations Research*, 40:1176–1185, 2013.

[21] Fomeni F.D. and Letchford A.N. A dynamic programming heuristic for the quadratic knapsack problem. *INFORMS Journal on Computing*, 26(1):173–182, 2014.

[22] Glover Fred. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[23] Glover Fred. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[24] Glover Fred., Jin-Kao Hao, and Yuning Chen. An evolutionary path relinking approach for the quadratic multiple knapsack problem. *Knowledge-Based Systems*, 92:23–34, 2016.

[25] Glover Fred and Laguna Manuel. *Tabu search*. Springer, 1999.

[26] Glover Fred, Laguna Manuel, and Martí Rafael. Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372, 2006.

[27] Dijkhuizen G. and Faigle U. A cutting-plane approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 69:121–130, 1993.

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

[29] Mathews GB. On the partition of numbers. *Proceedings of the London Mathematical Society*, 1(1):486–490, 1896.

[30] Christos H., Papadimitriou, and Steiglitz Kenneth. *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications, 1998.

[31] Holland John H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

[32] Akeb Hakim, Hifi Mhand, Mohamed Elhafedh, and Ould Ahmed Mounir. Local branching-based algorithms for the disjunctively constrained knapsack problem. *Computers & Industrial Engineering*, 60(4):811–820, 2011.

[33] Kellerer Hans, Pisinger David, and Pferschy Ulrich. *Knapsack problems*. Springer, 2004.

[34] Jin-Kao Hao and Yuning Chen. Iterated responsive threshold search for the quadratic multiple knapsack problem. *Annals of Operations Research*, 226(1):101–131, 2015.

[35] Jin-Kao Hao and Yuning Chen. Memetic search for the generalized quadratic multiple knapsack problem. *IEEE Transactions on Evolutionary Computation*, 20(6):908–923, 2016.

[36] Jin-Kao Hao and Yuning Chen. An iterated "hyperplane exploration" approach for the quadratic knapsack problem. *Computers Operations Research*, 77:226–239, 2017.

[37] Jin-Kao Hao. and Chen. Yuning. The bi-objective quadratic multiple knapsack problem, model and heuristics. *Knowledge-Based Systems*, 97:89–100, 2016.

[38] Keld Helsgaun. *An effective implementation of K-opt moves for the Lin- Kernighan TSP heuristic*. PhD thesis, Roskilde University. Department of Computer Science, 2006.

[39] Aarts Emile HL, Lenstra Jan Karel, and Vaessens Rob JM. A local search template. *Computers & Operations Research*, 25(11):969–979, 1998.

[40] Cowling Peter I and Keuthen Ralf. Embedded local search approaches for routing optimization. *Computers & operations research*, 32(3):465–490, 2005.

[41] ILOG IBM. Inc. cplex 12.5 user manual, 2012.

[42] David J., Gerhard J., Jr. Rader, and Woeginger. The quadratic 0–1 knapsack problem with series–parallel support. *Operations Research Letters*, 30(3):159–166, 2002.

[43] Orlin James. Contentment in graph theory: covering graphs with cliques. In *Indagationes Mathematicae (Proceedings)*, volume 80, pages 406–424. Elsevier, 1977.

[44] Bryant A Julstrom. Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. pages 607–614. ACM, 2005.

[45] Lee K., Park K., and Park S. An extended formulation approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 95:671–682, 1996.

[46] Kolesar and Peter J. A branch and bound algorithm for the knapsack problem. *Management Science*, 13(9):723–735, 1967.

[47] Eugene L. and Wood David E. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[48] Johnson L., Mehrotra A., and Nemhauser G.L. Min-cut clustering. *Mathematical programming*, 62(1-3):133–151, 1993.

[49] Létocart L., Nagih A., and Plateau G. Reoptimization in lagrangian methods for the 0-1 quadratic knapsack problem. *Computers and Operations Research*, 39:12–18, 2012.

[50] Hifi Mhand and Roucairol Catherine. Approximate and exact algorithms for constrained (un) weighted two-dimensional two-staged cutting stock problems. *Journal of combinatorial optimization*, 5(4):465–494, 2001.

[51] Hifi Mhand, Moussa Ibrahim, and Saadi Toufik. *Modèles de résolution approchée et efficace pour les problèmes des réseaux de transport et de télécommunication.* PhD thesis, Université de Picardie Jules Verne d'Amiens, 2015.

[52] Hifi Mhand and Michrafy Mustapha. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6):718–726, 2006.

[53] Hifi Mhand and Michrafy Mustapha. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & Operations Research*, 34(9):2657–2673, 2007.

[54] Al-Iedani Najat, Hifi Mhand, and Saadi Toufik. Neighborhood search-based heuristic for the k-clustering minimum biclique completion problem. volume 272, pages 639–643. IEEE, 2016.

[55] Al-Iedani Najat, Hifi Mhand, and Saadi Toufik. Recherche par voisinage pour le problème de k-clusters dans un graphe biparti. volume 206, pages 396–398, 2016.

[56] Al-Iedani Najat, Hifi Mhand, and Saadi Toufik. M'ethode hybride pour le problème de regroupement dans un graphe biparti. volume 117, 2017.

[57] Al-Iedani Najat, Hifi Mhand, and Saadi Toufik. A reactive search for the quadratic knapsack problem. volume 190. IEEE, 2017.

[58] Ergun Özlem, Ahuja Ravindra Kand, Orlin James B, and Punnen Abraham P. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, 2002.

[59] Bertsekas Dimitri P. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[60] Chaillou P., Hansen P., and Mahieu Y. Best network flow bound for the quadratic knapsack problem. in b. simeone, editor. *Combinatorial Optimization*, 1403:225–235, 1989.

[61] Hansen P. and Mladenović N. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449±467, 2001.

[62] James P., Kelly H., and Osman Ibrahim. *Meta-heuristics: theory and applications*. Springer, 1996.

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

[64] Jong Hyuk Park, Jeong-Jun Suh, Jong-Kuk Park, Shan Guo Quan, and Young Yong Kim. Multicast partition problem for smart home applications. In *Hybrid Information Technology, 2006. ICHIT'06. International Conference on*, volume 2, pages 284–291. IEEE, 2006.

[65] Siarry Patrick. *Métaheuristiques: Recuits simulé, recherche avec tabous, recherche à voisinages variables, méthodes GRASP, algorithmes évolutionnaires, fourmis artificielles, essaims particulaires et autres méthodes d'optimisation*. Editions Eyrolles, 2014.

[66] Magni Claudio Patrizio. Biclique completion problem: models and algorithms. *Master Project, Politecnico di Milano*, 2009.

[67] Shaw Paul. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998.

[68] Chrétienne Philippe, Gourdin Eric, Faure Nathalie, and Sourd Francis. Biclique completion problems for multicast network design. *Discrete Optimization*, 4(3):360–377, 2007.

[69] Hansen Pierre and Mladenović Nenad. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[70] Hammer P.L. and D.J. Rader Jr. Efficient methods for solving quadratic 0-1 knapsack problems. *INFOR*, 35:170–182, 1997.

[71] Battiti R. and Bertossi A. Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48(4):361–385, 1999.

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

[73] Anders Bo Rasmussen, Pisinger David, and Rune Sandvik. Solution of large-sized quadratic knapsack problems through aggressive reduction. pages 0107–8283, 2004.

[74] Peeters René. The maximum edge biclique problem is np-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.

[75] JMW Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, 17(3):200–207, 1970.

[76] Bellman Richard. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

[77] Battiti Robeto, Brunato Mauro, and Mascia Franco. *Reactive Search and Intelligent Optimization*, volume 45. Springer, 2008.

[78] Dasgupta Sanjoy, Papadimitriou Christos H, and Vazirani Umesh. *Algorithms*. McGraw-Hill, Inc., 2006.

[79] Kirkpatrick Scott, Vecchi MP, et al. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[80] Gualandi Stefano. k-clustering minimum biclique completion via a hybrid cp and sdp approach. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 87–101. Springer, 2009.

[81] Gualandi Stefano, Magni Claudio, and Maffioli Francesco. A branch-and-price approach to k-clustering minimum biclique completion problem. *International Transactions in Operational Research*, 20(1):101–117, 2013.

[82] Christoph Witzgall. Mathematical methods of site selection for electronic message systems (ems). *NASA STI/Recon Technical Report N*, 76:18321, 1975.

[83] Jiming Liu Xiao-Feng Xie. Mini-swarm for the quadratic knapsack problem. *Honolulu, HI, USA*, pages 190–197, 2007.