Metaheuristic approaches for solving packing problems: 0/1 multidimensional knapsack problem and two-dimensional bin packing problem as examples

Soukaina Laabadi

UNIVERSITÉ HASSAN II DE CASABLANCA
FACULTÉ DES SCIENCES AÏN CHOCK

Centre d'Etudes Doctorales
Sciences Fondamentales & Appliquées

Laboratoire des Mathématiques et Applications (LMA)

ECOLE NORMALE SUPERIEURE
UNIVERSITÉ HASSAN II DE CASABLANCA

THÈSE

En vue de l'obtention du

DOCTORAT

Présentée par : Soukaina LAABADI

En : Sciences Mathématiques et Informatique
Spécialité : Mathématiques Appliquées et Informatique

Titre:

Metaheuristic approaches for solving packing problems: 0/1 multidimensional knapsack problem and two-dimensional bin packing problem as examples

---

Soutenue publiquement le 07/03/2020
A *la Faculté des Sciences-Ain Chock* devant le Jury **:**

Pr. Abdelbaki ATTIOUI	PES, Ecole Normale Supérieure – UH2C	**Président**
Pr. Larbi AFIFI	PES, Faculté des sciences Ain Chock – UH2C	**Rapporteur**
Pr. Adil BELLABDAOUI	PH, ENSIAS – UM5 Rabat	**Rapporteur**
Pr. Abdelwahed NAMIR	PES, Faculté des sciences Ben M'sik – UH2C	**Rapporteur**
Pr. Badr ABOU EL MAJD	PH, Faculté des sciences – UM5 Rabat	**Examinateur**
Pr. Boujemâa ACHCHAB	PES, ENSA de Berrechid – UHI Settat	**Examinateur**
Pr. Mohamed ETTAOUIL	PES, Faculté des sciences et techniques – USMBA Fès	**Examinateur**
Pr. Mohamed NAIMI	PH, ENSA de Berrechid – UHI Settat	**Co-encadrant**
Pr. Hassan EL AMRI	PES, Ecole Normale Supérieure – UH2C	**Directeur de thèse**

Dedication

In memory of my parents "Fatima Merzouk and Omar Laabadi" with whom I would have liked to share this moment

Acknowledgements

It is my pleasure to acknowledge the roles of several individuals who supported me in elaborating my thesis. Firstly, I would like to express my deepest appreciation to my advisor Prof. Hassan El AMRI for his thoughtful guidance, warm encouragement, and support in my research efforts from the beginning to the end. Likewise, I am extremely grateful to my co-advisor Prof. Mohamed NAIMI for having always encouraged and followed my research activities. He guided me through the process, was always patient and encouraging, and offered advice that made this a better study. I am very appreciative for the many hours of his time he gave me.

Besides my advisors, I would like to express my deep sense of gratitude to Prof. Boujemâa ACHCHAB for his lasting belief in me. It was a great pleasure and honor to collaborate with him.

I am also deeply indebted to the rest of my dissertation committee: The chairman, Prof. Abdelbaki ATTIOUI, and the thesis reporters, Prof. Laarbi AFIFI, Prof. Adil BELLABDAOUI and Prof. Abdelwahd NAMIR for taking time to review my dissertation and providing valuable suggestions for my work. My sincere thanks also go to the thesis examiners, Prof. Badr ABOU EL MAJD, Boujemâa ACHCHAB and Prof. Mohamed ETTAOUIL for their insightful comments and their questions that incented me to widen my research from various perspectives.

These acknowledgments would not be complete without mentioning my colleagues at Mathematics and Applications Laboratory of Ecole Normale Supérieure - Casablanca. Last but not the least, I extend my gratitude to my family and close ones for their support, kindness, and affection that kept me going in achieving my career goals. Thanks for all your encouragement.

Abstract

The present thesis is about metaheuristic approaches for packing problems. Focusing our investigation on two different problems in terms of kind of assignment: The multidimensional knapsack problem (0/1 MKP) that belongs to the output value maximization type and the two-dimensional bin packing problem (2D-BPP) that falls to the input value minimization type. The 0/1 MKP occupies an important place in packing problems thanks to its theoretical and practical interest, followed by the 2D-BPP that has been also used to model various decision-making processes. These two basic problems arise in numerous real-world applications. They occur, for example, in the allocation of resources problems, scheduling problems, cutting problems as well as in transportation and logistics, to mention just a few.

The 0/1 MKP can be informally stated as a problem of packing a set of items that maximizes the profit of a multidimensional knapsack as much as possible, taking into consideration the limited capacity of its dimensions. The knapsack dimensions can be, for instance, the maximum weight that can be carried, the maximum height, or the maximum available volume that can accommodate items. On the other hand, the 2D-BPP is the problem of packing, without overlapping, a given set of small rectangular-shaped items into a minimum number of large identical rectangles (called bins) with the edges of the items parallel to those of bins.

The 0/1 MKP as well as the 2D-BPP belong to NP-hard optimization problems, which means there is no exact algorithm that can find an optimal solution for such problems in polynomial time. However, they can be resolved with approximate methods, especially heuristics and metaheuristics. These methods seek to find a trade-off between the solution quality and the consuming-time but without ensuring the optimality. This thesis proposes various metaheuristics for solving the two packing problems. We develop an improved genetic algorithm that solves efficiently the 0/1 MKP in reasonable runtime as well as a hybrid approach that incorporates a *k*-means clustering method in the genetic algorithm. Moreover, we make use of a recent metaheuristic belonging to the swarm intelligence algorithms, namely the crow search algorithm (CSA). Indeed, we adapt the CSA to the context of 2D-BPP by applying two different techniques, a binarization technique and a hybridization technique. The performance of the proposed metaheuristics is evaluated through extensive computational experiments by using the benchmark data of the two problems. Finally, we should note that the proposed metaheuristics could be applied to various optimization problems to compute approximate solutions.

Keywords: 0/1 Multidimensional knapsack problem, Two-dimensional bin packing problem, Optimization, Metaheuristics, Genetic algorithms, *k*-means method, Crow search algorithm, Binarization technique, Hybridization.

Résumé

La présente thèse porte sur les approches heuristiques pour résoudre les problèmes de chargement. Nous avons focalisé notre attention sur deux types de problèmes : Un problème de maximisation et un problème de minimisation. Pour le premier type, on a pris comme exemple le fameux problème de sac à dos multidimensionnel binaire (0/1 MKP). Tandis que pour le deuxième type, on a opté pour le problème de bin packing à deux dimensions (2D-BPP). Ces deux problèmes ont suscité l'attention de plusieurs chercheurs au regard d'autres problèmes de chargement et continuent d'être enrichis par une bibliographie assez étendue. Sur le plan pratique, ces deux problèmes connaissent un regain d'intérêt majeur, notamment sur le secteur industriel et économique. Ces deux problèmes ont modélisé maints problèmes réels, tel que le problème d'affectation des ressources, le problème d'ordonnancement des tâches, le problème de découpe, comme ils font l'objet d'un grand intérêt dans le transport et la logistique.

Etant donné un ensemble d'objets, chaque objet est caractérisé par une taille et un profit et un sac à plusieurs dimensions. Le problème 0/1 MKP consiste à sélectionner un sous-ensemble d'objets à charger dans le sac tout en maximisant son profit total et en respectant la capacité maximale de ses dimensions. Les dimensions du sac peuvent indiquer, par exemple, le poids maximal pouvant être transporté, la hauteur maximale ou le volume maximal disponible pouvant accueillir des objets. Tandis que le problème 2D-BPP est défini de la façon suivante: Etant donné un ensemble d'objets rectangulaires et un nombre illimité de rectangles identiques, dits bins, de dimensions plus larges que celles des objets. Le problème consiste à charger tous les objets dans un nombre minimum de bins, sans chevauchement et en respectant la capacité limitée de chaque bin utilisé.

Les deux problèmes appartiennent aux problèmes NP-difficiles de l'optimisation combinatoire. En d'autres mots, il n'existe pas un algorithme exact permettant de trouver une solution optimale en temps polynomial. En effet, le temps nécessaire pour trouver la solution optimale croit exponentiellement au fur et à mesure que la taille de l'instance du problème augmente. Cependant, nous pouvons utiliser des méthodes approchées, notamment les heuristiques et les métaheuristiques, pour trouver des solutions de bonne qualité en un temps raisonnable mais sans garantir l'optimalité. Nous proposons dans cette thèse diverses métaheuristiques pour résoudre les deux problèmes de chargement. Nous développons un algorithme génétique amélioré qui résout efficacement le problème 0/1 MKP en un temps raisonnable, ainsi qu'une approche hybride qui incorpore une méthode de classification des *k*-moyennes dans un algorithme génétique. De plus, nous concevons une variante d'une métaheuristique récente basée sur l'intelligence en essaim, à savoir l'algorithme de recherche des corbeaux. Ce dernier est binarisé afin qu'il soit adapté au contexte du problème du 2D-BPP en appliquant deux techniques différentes: Une technique de binarisation et une technique d'hybridation. La performance des métaheuristiques proposées est évaluée grâce à des instances standards qui

existent dans la littérature des deux problèmes. Finalement, on note que les métaheuristiques proposées peuvent être appliquées à divers problèmes d'optimisation pour le calcul des solutions approchées.

Mots-clés : Problème de sac à dos multidimensionnel, Problème de bin packing bidimensionnel, Optimisation, Métaheuristiques, Algorithmes génétiques, Méthode des *k*-moyennes, Algorithme de recherche des corbeaux, Technique de binarisation, Technique d'hybridation.

Synthèse

Introduction

Les problèmes de chargement jouent un rôle essentiel dans de nombreux secteurs d'activités. D'une manière générale, ils considèrent un ensemble d'objets à ranger dans un ou plusieurs conteneur(s) (aussi nommé(s) sac(s), bin(s), bande(s), entre autres) de dimensions fixes ou variables, tout en respectant un ensemble de contraintes. Ces problèmes interviennent dans la logistique, qui est leur domaine trivial, dans le cadre de chargement d'avions, de bateaux, etc. Ils interviennent également dans l'industrie, dans le cadre de découpe des matériaux ou l'ordonnancement de la production. On peut les rencontrer aussi dans les systèmes de télécommunications pour l'allocation des fréquences, ou encore dans les systèmes informatiques dans le cadre des grilles de calcul ou d'affectation des ressources. La variété des applications réelles des problèmes de chargement a conduit à des diverses classifications (voir l'article de (Dyckhoff, 1990)). A ce propos, une classe plus générale connue sous le nom de "Cutting and Packing" (C&P) ou "Découpe et Chargement", a été développée par (Wäscher, et al., 2007). En effet, la classe C&P a été développée en combinant deux critères : La quantité disponible des conteneurs pour ranger les objets et l'assortiment des objets. Une quantité suffisamment grande des conteneurs conduit à un problème de minimisation tandis qu'une quantité limitée des conteneurs donne lieu à un problème de maximisation de profit. Dans la présente thèse, on a considéré deux types de problèmes qui découlent de la classe C&P (voir Figure 1) : Le problème de sac à dos (problème de maximisation) et le problème de bin packing (problème de minimisation). Pour le premier problème, on a considéré la variante dans laquelle le sac dispose de plusieurs dimensions « le sac à dos multidimensionnel ». Alors que pour le deuxième problème, on s'est intéressé à la variante dans laquelle les conteneurs aussi bien que les objets à ranger ont des formes rectangulaires « le bin packing bidimensionnel ».

Les deux problèmes considérés comprennent des enjeux financiers : Un bon rangement des objets permet de réduire les coûts d'exploitation et d'augmenter les rendements. C'est pourquoi ces problèmes intéressent aussi bien les chercheurs que les chefs d'entreprises.

Le problème de sac à dos et le problème de bin packing sont des problèmes classiques de l'optimisation combinatoire qui consiste à trouver une meilleure solution, dite solution optimale ou optimum, parmi un ensemble fini (souvent très grand) de solutions possibles, dites aussi solutions réalisables. Le nombre d'alternatives augmente exponentiellement en fonction de la taille du problème considéré, et du coup la recherche de la meilleure alternative dans l'espace des solutions demeure une tâche fastidieuse. En effet, la plupart des problèmes d'optimisation combinatoire sont connus pour être très difficiles à résoudre bien qu'ils soient faciles à exprimer, un meilleur exemple est celui des deux problèmes considérés dans cette thèse. Une classe des problèmes difficiles particulièrement étudiée est celle des problèmes NP-difficiles,

dans laquelle l'énumération exhaustive des solutions est envisageable en un temps exponentiel ou pire qu'exponentiel selon la taille du problème étudié. Le problème de sac à dos multidimensionnel et le problème du bin packing bidimensionnel sont prouvés d'être des problèmes NP-difficiles selon (Johnson & Garey, 1979).

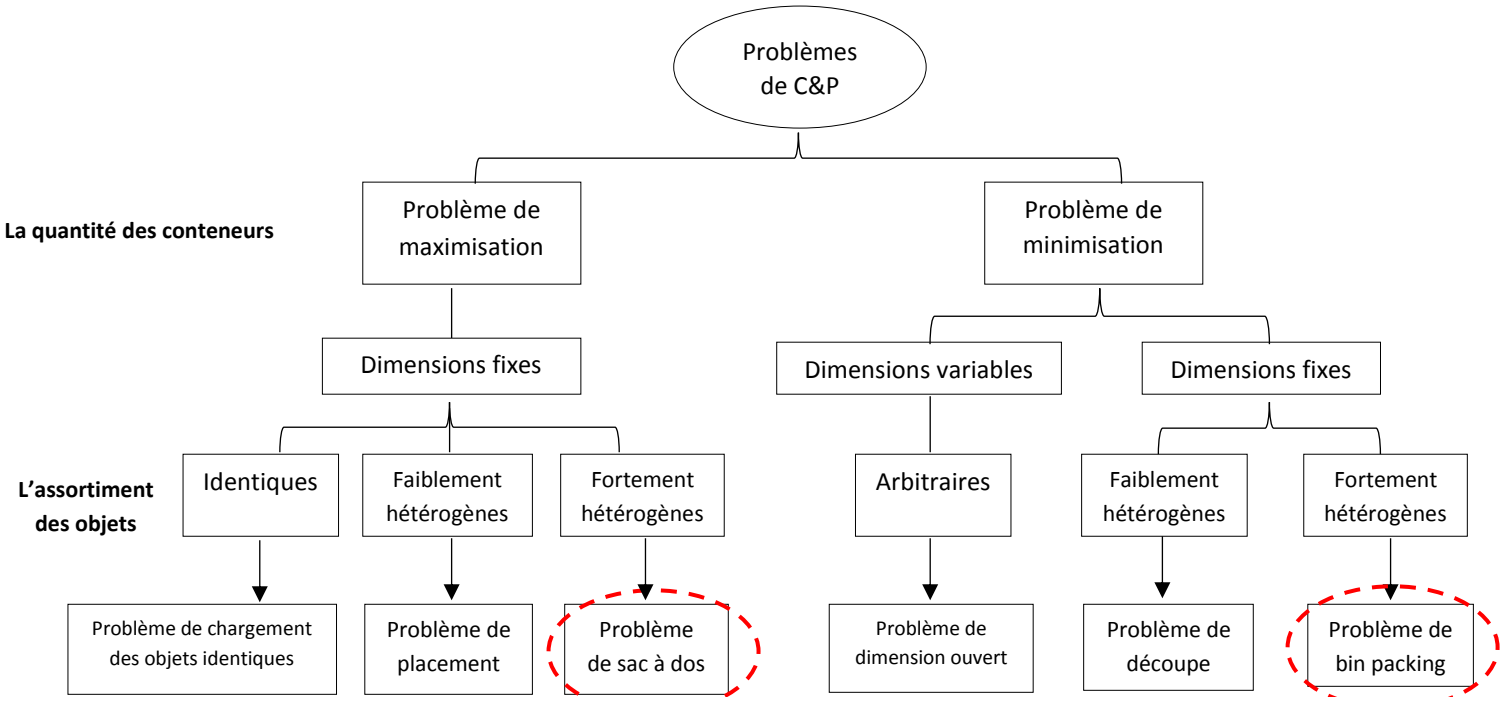


Figure 1. Les types de problèmes appartenant à la classe C&P

La résolution des problèmes NP-difficiles peut être abordée par des méthodes exactes ou par des méthodes approchées. Les méthodes exactes consistent à parcourir tout l'espace de recherche des solutions afin d'en extraire une solution optimale. Elles sont efficaces pour des problèmes de taille modérée. Mais, dès que la taille des problèmes augmente, les méthodes exactes deviennent très coûteuses en termes du temps de calcul. On est donc amené souvent à une résolution approchée qui tente à chercher une solution de bonne qualité, mais qui n'est pas nécessairement optimale, dans un laps de temps raisonnable : C'est ce que font les heuristiques et les métaheuristiques. Une heuristique est un algorithme désigné à un problème donné. Tandis qu'une métaheuristique est un schéma algorithmique qui peut être généralisé à plusieurs problèmes d'optimisation combinatoires tout en appliquant des modifications mineures. En effet, les métaheuristiques utilisent des stratégies indépendantes de la structure du problème auquel on les applique. Ces stratégies ont pour but de guider la recherche dans l'espace des solutions visant à l'explorer le plus efficacement possible et à intensifier la recherche dans les zones qui peuvent probablement contenir les meilleures solutions, dites zones prometteuses. Une bonne métaheuristique est celle qui oscille entre l'exploration de l'espace de recherche (la stratégie de diversification) et l'exploitation de ses zones prometteuses (la stratégie d'intensification). Il existe un grand nombre de métaheuristiques allant de schémas très simples

qui mettent en œuvre des processus de recherche basiques comme la descente (Cauchy, 1847) ou la méthode de recherche locale; à des schémas beaucoup plus complexes et minutieuses avec des processus inspirés de la nature. Comme exemples de métaheuristiques sophistiquées, on peut citer le recuit simulé qui est inspiré d'un processus métallurgique (Kirkpatrick, et al., 1983), les algorithmes génétiques qui sont inspirés du processus de reproduction naturelle (Holland, 1975), l'algorithme de pollinisation des fleurs qui est inspiré, comme son nom l'indique, du principe de la fécondation des plantes à fleurs par le transport du pollen (Yang, 2012). On peut citer également les algorithmes basés sur l'intelligence en essaim qui s'inspirent de l'auto-organisation et le comportement collectif de certaines espèces comme l'algorithme d'optimisation par essaim de particules (Eberhart & Kennedy, 1995), l'algorithme de colonies de fourmis (Dorigo, et al., 1996), l'algorithme de colonies d'abeilles (Karaboga, 2005) ainsi que l'algorithme de recherche des corbeaux (Askarzadeh, 2016).

Contributions

Les contributions réalisées au cours de cette thèse concernent la résolution approchée des deux problèmes de chargement : Le problème de sac à dos multidimensionnel et le problème de bin packing bidimensionnel. Une analyse comparative des résultats expérimentaux est effectuée avec des algorithmes de nature similaire existants dans la littérature.

Première contribution : Nous avons développé un algorithme génétique sexuel pour résoudre le problème de sac à dos multidimensionnel (Laabadi, et al., 2019a). Dans un premier lieu, nous avons amélioré une stratégie de sélection introduite par (Varnamkhasti & Lee, 2012a), qui prend en considération le sexe des chromosomes parents dans la phase de construction des couples et qui est appelée « la sélection sexuelle ». Cette sélection divise la population des chromosomes en deux groupes ; groupe des femelles et groupe des mâles. Le groupe des femelles qui va participer à la reproduction, est choisi par une méthode de sélection traditionnelle « sélection par tournoi ». Tandis que leurs partenaires (chromosomes mâles) sont choisis en se basant sur des critères de préférence, tels que la distance de Hamming entre un chromosome mâle et un chromosome femelle, la fonction fitness ou les gènes actifs (c-à-d les gènes ayant la valeur 1) des chromosomes mâles. Cette stratégie est définie dans l'espace génotypique. Dans cette thèse, nous avons adapté cette stratégie de sélection sexuelle à l'espace phénotypique en tirant profit de la structure du problème considéré afin d'améliorer la qualité des solutions. Dans un second lieu, nous avons proposé deux nouvelles versions d'un opérateur de croisement proposé dans le cadre du sac à dos multi-objectif par (Aghezzaf & Naimi, 2009). Cet opérateur vise à préserver les gènes en commun des chromosomes parents en se basant sur deux grandes étapes. Dans la première étape, le chromosome enfant hérite les gènes similaires de ces parents. Ensuite, les gènes non similaires sont traités dans la deuxième étape en utilisant une fonction de préférence adaptée au contexte multi-objectif pour choisir quels gènes de quel parent vont être hérités par le chromosome enfant. Notre apport a été au niveau de la deuxième étape. La première version est une adaptation de cet opérateur au contexte du sac à dos

multidimensionnel mono-objectif. En revanche, dans la deuxième version, on a injecté l'aspect aléatoire dans l'opérateur de croisement dans l'espérance de diversifier la recherche dans l'espace des solutions. Finalement, on a combiné la stratégie de sélection sexuelle proposée avec l'opérateur de croisement (la première version ou la deuxième version) dans un seul algorithme génétique afin de bénéficier de la synergie remarquée entre ces deux opérateurs génétiques. Cette synergie a remédié au problème classique des algorithmes génétiques qui est la convergence prématurée tout en permettant une exploration plus efficace de l'espace de recherche des solutions.

Deuxième contribution : On a proposé une approche hybride en combinant une méthode de classification, plus précisément la méthode des *k*-moyennes, avec un algorithme génétique (Laabadi, et al., 2018c). Cette approche a été appliquée au problème de sac à dos multidimensionnel. En effet, une méthode de classification, en l'occurrence la méthode des *k*-moyennes, consiste à grouper des observations (numériques, textuelles,…) dans des groupes de telle sorte que les individus dans un même groupe se ressemblent le plus possible et les individus dans des groupes différents se démarquent le plus possible. Partant de cette idée, nous avons utilisé la méthode des *k*-moyennes pour la sélection des parents chromosomes en fixant k à 2. Premièrement, on divise la population initiale en deux groupes de chromosomes en utilisant la méthode des *k*-moyennes. Deuxièmement, on construit les couples de la façon suivante : Le groupe ayant moins de chromosomes participera tout entier à la reproduction. Pour chaque chromosome de ce groupe, on choisit son partenaire parmi les chromosomes du deuxième groupe résultant, en leur faisant une compétition grâce à la méthode de sélection par tournois. Il est clair que la méthode des *k*-moyennes va permettre de créer une diversification dans la population puisqu'elle choisit des parents de caractéristiques distantes. De plus, la sélection par tournois va garantir une bonne exploitation de l'espace des solutions. Par conséquent, notre stratégie de sélection basée sur la méthode des *k*-moyennes, sera en mesure d'assurer un bon équilibre entre la diversification et l'intensification pour un algorithme génétique.

Troisième contribution : Pour la résolution du problème de bin packing bidimensionnel, nous nous sommes orientés vers les métaheuristiques basées sur l'intelligence en essaim, plus précisément l'algorithme de recherche des corbeaux. Cette métaheuristique récente a été introduite initialement par (Askarzadeh, 2016). Dans cet algorithme, l'auteur s'est inspiré de l'intelligence des corbeaux pour cacher leur nourriture et la protéger d'une éventuelle pillerie, ainsi que leur capacité de mémorisation pour retrouver leur cachette de nourriture. L'algorithme de recherche des corbeaux a été destiné premièrement aux problèmes d'optimisation continue. Dans notre thèse, nous avons binarisé cette métaheuristique par le biais d'une transformation Sigmoid (Laabadi, et al., 2020), afin qu'elle soit adaptée aux problèmes d'optimisation binaire en général et au problème de bin packing bidimensionnel en particulier.

Quatrième contribution : Nous avons proposé une autre façon d'étendre l'algorithme de recherche des corbeaux aux problèmes d'optimisation binaire. Ce procédé est basé sur la combinaison de l'algorithme de recherche des corbeaux avec les algorithmes génétiques (Laabadi, et al., 2019b). L'idée est de préserver les étapes principales de l'algorithme de recherche des corbeaux tout en utilisant des opérateurs génétiques : L'opérateur de sélection, de croisement et de mutation. Ces opérateurs ont pour but de binariser l'algorithme de recherche des corbeaux, mais aussi de balancer entre la stratégie de diversification et celle d'intensification au cours du déroulement de l'approche proposée. Un autre avantage de cette approche hybride est qu'elle peut être appliquée à tous types de problèmes d'optimisation, plus particulièrement ceux qui sont binaires.

Organisation du manuscrit

Hormis l'introduction et la conclusion générale, ce manuscrit est divisé en deux grandes parties : La première partie est consacrée au problème de sac à dos multidimensionnel et la deuxième partie concerne le problème de bin packing bidimensionnel.

Première Partie : Le problème de sac à dos multidimensionnel

Cette partie compte quatre chapitres :

Chapitre 1. Définition, variantes et applications : Dans ce chapitre nous commençons par une description du problème et nous citons ses variantes les plus connues en donnant une définition générale pour chacune. Nous décrivons ensuite quelques applications réelles de ce problème ainsi que ses variantes.

Chapitre 2. Méthodes de résolution: Ce chapitre aborde brièvement quelques méthodes exactes connues pour la résolution du problème de sac à dos multidimensionnel. Il présente aussi un état de l'art très récent des méthodes heuristiques développées pour la résolution de ce problème.

Chapitre 3. Algorithme génétique sexuel amélioré : Dans ce chapitre, nous décrivons d'abord des nouvelles variantes proposées d'une stratégie de sélection sexuelle et d'un opérateur de croisement, avant de présenter le processus complet de l'algorithme génétique sexuel proposé pour résoudre le problème en cours.

Chapitre 4. Algorithme génétique basé sur la méthode des k-moyennes : Dans ce chapitre, nous détaillons un nouvel algorithme génétique combiné avec la méthode des *k*-moyennes. Dans un premier lieu, nous exposons l'état de l'art des algorithmes hybrides appliqués au problème en cours. Ensuite, nous décrivons la démarche de notre approche hybride.

Deuxième Partie : Problème de bin packing bidimensionnel

Cette partie comporte trois chapitres :

Chapitre 5. Définition, variantes et méthodes de résolution : Dans ce chapitre nous commençons par une description du problème de bin packing bidimensionnel en mettant l'accent sur la variété de contraintes pratiques qui restreignent celui-ci. Puis nous citons les typologies les plus fréquentes et quelques modèles d'applications. Nous présentons ensuite les bornes inférieures les plus connues ainsi que les méthodes de résolution de ce problème.

Chapitre 6. Algorithme binaire de recherche des corbeaux : Ce chapitre présente un état de l'art des applications de l'algorithme de recherche des corbeaux. Ensuite, il décrit la version binaire de l'algorithme de recherche des corbeaux dédiée au problème du bin packing bidimensionnel.

Chapitre 7. Algorithme de recherche des corbeaux hybridé avec un algorithme génétique : Dans ce chapitre nous détaillons une nouvelle approche hybride qui incorpore des opérateurs génétiques dans l'algorithme de recherche des corbeaux. Dans un premier temps, nous présentons l'état de l'art des algorithmes hybrides appliqués au problème du bin packing bidimensionnel. Ensuite, nous expliquons notre approche hybride et ses composantes.

Contenu des chapitres

Chapitre 1 :

Le problème du sac à dos multidimensionnel à variables binaires est l'un des problèmes d'optimisation académiques les plus importants, il est noté par 0/1 MKP (pour 0/1 Multidimensionnal Knapsack Problem). Le problème 0/1 MKP peut être défini de la façon suivante : Etant donné un conteneur, appelé « sac à dos », de dimensions connues (volume, poids, quantité…) et un ensemble fini d'objets. Chaque objet est doté d'un profit et d'une taille relative aux dimensions du sac à dos. Le problème 0/1 MKP consiste à sélectionner un sous-ensemble d'objets qui maximise le profit total du sac sans dépasser la capacité limitée de chacune de ses dimensions. Ce problème a plusieurs variantes dont chacune présente ses propres spécificités et contraintes pratiques. Les variantes de ce problème peuvent être catégorisées en deux classes selon la disponibilité de l'information (certaine ou incertaine):

- La classe des problèmes de sac à dos multidimensionnel déterministes dans laquelle on dispose d'une information exacte sur les caractéristiques des objets ainsi que les dimensions du sac. Celle-ci inclut les variantes suivantes : Le problème de sac à dos multidimensionnel multiple dans lequel plusieurs sacs sont disponibles. Le problème de sac à dos multidimensionnel à choix multiple, dans lequel les objets sont classés dans des groupes disjointes, consistant à maximiser le profit du sac sous deux types de contraintes linéaires : Les contraintes de capacité du sac et la contrainte de sélection d'au plus un objet de chaque groupe. On peut citer aussi le problème de sac à dos multidimensionnel multi-objectif qui prend en considération, lors de la sélection des objets, plusieurs critères (ou objectifs) à optimiser.

- La classe des problèmes de sac à dos multidimensionnel non-déterministes dans laquelle l'information est incertaine. Soit l'information est vague, c'est à dire qu'on peut avoir

plusieurs possibilités pour une caractéristique donnée du problème. Ce cas donne lieu à une variante définie dans un environnement flou (Khalili-Damghani & Taghavifard, 2011), appelée « le problème de sac à dos multidimensionnel flou ». Soit l'information suit une distribution de probabilité de paramètres connus, ce qui fait naître une variante probabiliste (Kunikazu & Prékopa, 2012), nommée « le problème de sac à dos multidimensionnel stochastique ». Dans cette variante une contrainte probabiliste est jointe au modèle mathématique et qui représente le degré de la violation des capacités du sac vis-à-vis d'un certain seuil.

L'importance académique du problème 0/1 MKP est liée, d'une part, au fait qu'il peut être utilisé par des chercheurs pour évaluer la performance de leurs algorithmes, et d'autre part, à sa structure simple qui peut être utilisée en tant que sous problème pour des problèmes complexes d'optimisation. En outre, ce problème possède une importance pratique puisqu'il peut être utilisé comme modèle pour formuler mathématiquement des problèmes d'optimisation concrets et issus de différents domaines (Laabadi, et al., 2018a). On le retrouve essentiellement dans l'économie, l'informatique, les systèmes de télécommunications et réseaux, ou encore dans le génie civil. Plus précisément, le problème 0/1 MKP permet de modéliser des problèmes d'optimisation où il s'agit de maximiser un profit tout en ne disposant que des ressources limitées. Nous citons à titre d'exemples les problèmes d'optimisation réels suivants :

- Le problème de vente aux enchères : La vente aux enchères est une vente publique caractérisée par l'attribution des propriétés aux acheteurs les plus offrants, dans le but de maximiser le revenu du commissaire-priseur. Une extension de ce problème est la vente aux enchères multi-unités dans laquelle une propriété peut être disponible en plusieurs unités. (Pfeiffer & Rothlauf, 2007) ont formulé ce problème mathématiquement grâce au modèle du problème 0/1 MKP. En fait, les enchérisseurs correspondent aux objets à ranger dans un sac et les quantités limitées de chaque propriété représentent les contraintes de capacité du sac. De plus, le profit de chaque objet correspond au prix offert par l'enchérisseur, tandis que la taille de chaque objet correspond au nombre d'unités demandées d'une propriété donnée.

- Le problème de budgétisation du capital : Il consiste à sélectionner les projets d'investissement les plus rentables pour une compagnie. (Khalili-Damghani & Taghavifard, 2011) ont été intéressé par une variante floue de ce problème dans laquelle l'information, concernant les quantités des ressources nécessaires pour l'implémentation d'un projet donné, est ambiguë. Afin de modéliser ce problème, les auteurs ont utilisé le problème du 0/1 MKP flou. En effet, les objets sont les projets et les dimensions du sac sont les quantités disponibles de chaque type de ressources (humaines, financières, matérielles…). Le profit de chaque objet correspond aux bénéfices apportés par chaque projet, tandis que la taille de chaque objet correspond à la quantité demandée de chaque type de ressources pour la réalisation d'un projet.

- Le problème d'allocation des ressources avec demandes stochastiques dans un système distribué : Il consiste à attribuer des ressources à des utilisateurs d'un système distribué. Une extension de ce problème se produit lorsque les demandes des utilisateurs sont incertaines. (Chen, et al., 2012) ont considéré que les demandes suivent des distributions de probabilité ayant des paramètres connus. Par conséquent, ils ont modélisé ce problème en utilisant le problème 0/1 MKP stochastique dans lequel une contrainte probabiliste est jointe aux contraintes de capacité. En effet, les utilisateurs correspondent aux objets et les différents types de ressources correspondent aux dimensions. Chaque utilisateur peut ainsi profiter d'une utilisation efficace du système tout en demandant des quantités de ressources de différents types, ce qui correspond respectivement au profit de chaque objet et sa taille dans chaque dimension du sac. L'objectif consiste donc à satisfaire les demandes des utilisateurs de façon à maximiser la performance du système distribué.

- Le problème d'optimisation de consommation énergétique dans un système multi-processeur MPSoC: Il consiste à minimiser l'énergie consommée par les applications actives dans un MPSoC. Sachant qu'une application active peut avoir plusieurs implémentations possibles, ce problème a été modélisé par (Ykman-Couvreur, et al., 2001) grâce au problème 0/1 MKP à choix multiple. En effet, chaque application active représente une classe contenant ses implémentations possibles. Les applications actives correspondent aux classes disjointes et les implémentations possibles sont considérées comme des objets. Chaque implémentation d'une application donnée consomme une énergie et nécessite une quantité de ressources de différents types (espace mémoire, processeurs…), ce qui représente respectivement le profit et la taille des objets. La fonction objectif dans ce problème est à minimiser d'origine, mais les auteurs l'ont converti en une fonction de maximisation qui vise à sélectionner, pour chaque application active, une implémentation dont la différence entre son énergie consommée et la consommation maximale d'énergie doit être aussi grande que possible.

- Le problème d'allocation des fréquences dans un réseau de radio cognitive (RC) : Un réseau fonctionnant avec la technologie RC permet de gérer dynamiquement un spectre. Le principe de fonctionnement d'un tel réseau est le suivant : Un utilisateur secondaire pourra à tout moment accéder à des bandes de fréquence qu'il trouve libres (*i.e.* non occupées par l'utilisateur primaire possédant une licence sur cette bande). Le problème d'allocation des fréquences dans un réseau RC consiste à maximiser l'utilisation des bandes de fréquence disponibles dans le spectre en prenant en compte les interférences avec d'autres utilisateurs. (Song, et al., 2008) ont formulé ce problème grâce au problème 0/1 MKP multiple de la façon suivante. Les utilisateurs de la RC sont considérés comme les objets à sélectionner et les bandes des utilisateurs primaires sont considérées comme les sacs disponibles, tandis que la largeur de chaque bande ainsi que son seuil d'interférence correspondent aux dimensions du sac. Chaque utilisateur crée un débit dans le spectre, ce qui correspond au profit d'un objet. Il transmet des fréquences à travers les bandes des utilisateurs primaires

et génère une interférence entre les autres utilisateurs, ce qui correspond à la taille d'un objet relative à chaque dimension (Largeur du bande et seuil d'interférence).

- Le problème de maintenance immobilière : La maintenance immobilière a vocation de traiter les équipements intérieurs, les équipements techniques et les ouvrages extérieurs afin de les remettre dans un bon état. Pour cela, un plan d'action est nécessaire. Ainsi, le problème de maintenance immobilière consiste à trouver un plan pluriannuel optimal à réaliser dans un délai limité, en respectant un certain budget. (Taillandier, et al., 2017) ont formulé ce problème grâce au problème 0/1 MKP multi-objectif dans lequel les actions sont les objets à sélectionner, et les années nécessaires pour réaliser les actions représentent les dimensions du sac. Chaque action a un coût d'exécution à chaque année, et un score qui correspond à l'état d'un immobilier (concernant un critère donné) qui doit être maximisé pour indiquer qu'il est en bon état, présentant respectivement la taille d'un objet dans chaque dimension et son profit.

La bibliographie de la famille des problèmes de sac à dos multidimensionnel et leurs applications réelles est très riche. Dans ce chapitre nous avons défini le problème 0/1 MKP. Par la suite, nous avons décrit ses variantes les plus fréquentes. Puis, nous avons exposé quelques champs d'application de certaines variantes du problème.

Chapitre 2 :

Le fait que le problème 0/1 MKP soit un problème-test pour les nouveaux algorithmes et le fait qu'il soit un problème-modèle pour la formulation de nombreux problèmes issus du monde réel, ont stimulé l'apparition d'une panoplie de méthodes pour sa résolution. Ces dernières diffèrent selon leur performance en termes de qualité des solutions et du temps de calcul. En réalité, on peut discerner toutes ces méthodes en deux classes principales. La première contient les algorithmes exacts qui assurent l'obtention de la solution optimale sauf qu'elles consomment un temps très couteux pour traiter les instances larges du problème considéré. La deuxième est celle des méthodes approchées qui produisent des solutions de bonne qualité en un temps raisonnable, mais sans garantir de l'optimalité.

C'est à partir des années soixante que les méthodes exactes commençaient à susciter l'intérêt des théoriciens en recherche opérationnelle. Toutefois, les méthodes exactes sont relativement moins fréquentes dans la littérature puisqu'elles sont souvent impraticables lorsque la taille de l'instance à traiter (nombre d'objets, nombre de dimensions, etc.) est très large. L'une des premières méthodes exactes qui ont été développées pour résoudre le problème 0/1 MKP est la programmation dynamique proposée par (Gilmore & Gomory, 1966). Une année après, de nouvelles variantes de la programmation dynamique ont été suggérées par (Green, 1967). Par la suite, la méthode de résolution par séparation et évaluation avait été développée pour concurrencer la programmation dynamique dans la résolution du problème 0/1 MKP ((Thesen, 1975), (Shih, 1979), (Gavish & Pirkul, 1985)). Jusqu'à cette dernière décennie, la méthode

séparation et évaluation présente un axe d'intérêt pour la résolution exacte du problème. On cite la procédure de recherche par séparation et évaluation du (Boussier, et al., 2010), qui a résolu des instances contenant jusqu'à 500 objets et 30 contraintes. Cependant, leur méthode a consommé des heures à des jours pour trouver la solution optimale particulièrement pour les instances les plus larges.

En revanche, il existe un arsenal important d'algorithmes approximatifs qui ont été appliqués au problème 0/1 MKP (Laabadi, et al., 2018a); ce qui est dû au NP-difficulté de ce problème. Ainsi, les méthodes de résolution approchée qui sont dédiées à ce problème sont aussi classées à leur tour en deux catégories: Les heuristiques qui sont conçues spécifiquement au problème 0/1 MKP et les métaheuristiques qui peuvent être adaptées à une large variété de problèmes d'optimisation en appliquant des modifications mineures. Les heuristiques comprennent notamment deux types considérés comme les plus fréquents pour la résolution du problème 0/1 MKP:

- Les algorithmes gloutons: Ils consistent à construire une solution pas à pas sans revenir sur les décisions précédentes. Ils effectuent à chaque étape le choix qui semble le meilleur en espérant obtenir un optimum global. Parmi ces algorithmes on retrouve les algorithmes proposés par (Kochenberger, et al., 1974), (Toyoda, 1975) et (Loulou & Michaelides, 1979)). Un autre type qui se base sur la résolution itérative du problème 0/1 MKP est l'algorithme glouton dual qui a été introduit par (Senju & Toyoda, 1968) consistant à remplir le sac par tous les objets (ce qui donne bien évidemment une solution irréalisable), puis retirer pas à pas les objets qui violent les contraintes de capacité du sac jusqu'à l'obtention d'une solution réalisable.

- Les heuristiques basées sur la relaxation linéaire, la relaxation Lagrangienne, ou encore la relaxation surrogate: Ces heuristiques essaient de construire une solution réalisable pour le problème 0/1 MKP à partir de l'analyse de la solution de sa relaxation linéaire ((Balas & Martin, 1980), (Lee & Guignard, 1988), (Fleszar & Hindi, 2009), et (Hanafi & Wilbaut, 2011)) ou en utilisant sa relaxation Lagrangienne ((Magazine & Oguz, 1984), (Atılgan & Nuriyev, 2012), (Yoon, et al., 2012), et (Yoon & Kim, 2013)), ou encore en appliquant la relaxation surrogate ((Pirkul, 1987), (Fréville & Plateau, 1996), (Montaña, et al., 2008), et (Boyer, et al., 2009)). Ces deux dernières relaxations peuvent être combinées pour donner naissance à une nouvelle relaxation désignée sous le terme « relaxation composite » qui a été introduite initialement par (Greenberg & Pierskalla, 1970).

Devant la limite des résultats obtenus par les heuristiques, un nombre important de métaheuristiques ont été proposées pour résoudre le problème 0/1 MKP. Ces algorithmes sont plus complets et beaucoup plus complexes (temporellement et spatialement) qu'une simple heuristique et s'inspirent généralement des processus naturels qui incluent plusieurs stratégies visant à guider la recherche vers les meilleures solutions. Ces méthodes sont partagées en deux

classes : Les métaheuristiques à base de solution unique et celles à base de solutions multiples. Les méthodes à base de solution unique se basent généralement sur une recherche par voisinage (ou recherche locale). La plupart de ces méthodes possèdent des mécanismes pour s'échapper des optima locaux. Parmi les métaheuristiques à solution unique appliquées au problème 0/1 MKP, on cite les algorithmes basés sur la recherche tabou qui ont été proposés par ((Dammeyer & Voss., 1993), (Glover & Kochenberger, 1996), (Løkketangen & Glover, 1996), (Niar & Freville, 1997), (Vasquez & Hao, 2001), et (Vasquez & Vimont, 2005)). On cite également les algorithmes basés sur la méthode de recherche à voisinage variable ((Puchinger & Raidl, 2005), (Puchinger, et al., 2006), (Puchinger & Raidl, 2008), (Hanafi, et al., 2009), et (Tasgetiren, et al., 2015)). Finalement, on cite les approches à base de l'algorithme du recuit simulé qui a été utilisé dans le cadre du 0/1 MKP depuis les années quatre-vingts par (Drexl, 1988), et qui n'a reçu l'attention des auteurs que dernièrement par ((Leung, et al., 2012) et (Gupta & Arora, 2015)). Quant aux méthodes à base de solutions multiples, elles se basent sur une recherche globale sur tout l'espace d'état et partent d'une population de solutions. Elles ont été et continuent d'être l'axe d'intérêt d'une grande gamme de chercheurs. Parmi les méthodes à base de solutions multiples qui sont fréquemment utilisées pour la résolution du 0/1 MKP, on trouve les algorithmes génétiques qui s'inspirent de l'évolution Darwinienne, partant d'une population de solutions et appliquant ainsi des opérateurs génétiques pour l'amélioration de la qualité des solutions générées jusqu'à un nombre fini de générations. Dans ce sens, on peut citer les travaux de ((Chu & Beasley, 1998), (Varnamkhasti & Lee, 2012a), (Yang, et al., 2013), et (Martins, et al., 2014)). L'autre type de méthodes à base de solutions multiples concerne les algorithmes d'intelligence en essaim qui s'inspirent à leur tour du comportement collectif de certaines espèces. On cite ainsi les algorithmes d'optimisation des essaims particulaires développés par ((Wang, et al., 2009), (Langeveld & Engelbrecht, 2012), et (Lin, et al., 2016)), et les algorithmes de colonies de fourmis proposés par ((Leguizamon and Michalewicz 1999), (Fidanova, 2002), et (Fingler, et al., 2014)).

Ce chapitre a été consacré aux méthodes de résolution du problème 0/1 MKP. Nous avons présenté un état de l'art qui survole les travaux anciens autant que les plus récents. En faisant un tour d'horizon des méthodes exactes et des méthodes approchées, nous avons remarqué que peu de travaux portent sur la résolution exacte, tandis que la résolution approchée est une technique fréquemment rencontrée que ce soit en utilisant les heuristiques ou les métaheuristiques. Dans le chapitre suivant, nous exposerons notre premier algorithme proposé dans le cadre du problème 0/1 MKP.

Chapitre 3 :

Les algorithmes génétiques (AGs) ont été initialement développés par (Holland, 1975). Après la publication du livre intitulé "*Genetic Algorithms in Search, Optimization and Machine Learning*" écrit par (Goldberg, 1989), les algorithmes génétiques deviennent de plus en plus populaires. Une large gamme de chercheurs a tiré profit de l'implémentation simple et la

capacité des AGs d'être adaptés à n'importe quel type d'optimisation pour résoudre les problèmes rencontrés dans leurs domaines. En effet, Les AGs sont considérés comme un outil performant d'optimisation fondé sur le processus d'évolution de Darwin (Darwin, 1913). L'évolution Darwinienne stipule que les espèces animales ou végétales doivent s'adapter aux variations de leur environnement pour survivre, quitte à subir des modifications biologiques et/ou comportementales. Ceux qui survivent sont ceux qui produisent des descendants, ce qui garantissent la transmission des meilleures caractéristiques à travers les générations. Le fonctionnement des AGs est simple. On part d'une population initiale d'individus, dits *chromosomes*, qui sont arbitrairement choisis. Chaque chromosome représente une solution potentielle du problème traité. On évalue la performance des chromosomes selon une fonction appelée *fonction d'adaptation* (ou *fonction fitness*). Sur la base de leur performance, on crée une nouvelle population de solutions qui est constituée de *chromosomes enfants* tout en appliquant les opérateurs génétiques suivants :

- L'opérateur de sélection : Il consiste à choisir les chromosomes les mieux adaptés afin qu'ils participent à la reproduction des chromosomes enfants. Cette stratégie respecte le principe de la théorie de Darwin en préservant les chromosomes les plus adaptés. Il existe plusieurs techniques de sélection dont la plupart sont basées sur le degré d'adaptation des chromosomes à un problème donné. Nous citons quelques exemples d'opérateurs de sélection : *La sélection par tournois* qui choisit aléatoirement deux chromosomes ou plus, et leur fait une compétition selon leur degré d'adaptation et choisit finalement le mieux adapté. *La sélection par roulette* qui sélectionne les chromosomes selon une probabilité proportionnelle à leur degré d'adaptation. *La sélection par troncature* qui choisit d'une manière déterministe une proportion (entre 10% et 50%) de chromosomes qui vont produire des chromosomes enfants. On trouve aussi *la sélection stochastique universelle* qui repose sur l'utilisation d'un segment linéaire subdivisée à N fragments (avec N est la taille de la population) ayant une taille proportionnelle au degré d'adaptation du chromosome correspondant. Soit Nb le nombre de chromosomes qui vont participer au croisement, la sélection des chromosomes est ainsi désignée par un ensemble de Nb pointeurs équidistants placés au long du segment. La distance entre les pointeurs est égale à $1/Nb$, tandis que la position du premier pointeur est générée aléatoirement dans l'intervalle $[0;\ 1/Nb]$.

- L'opérateur de croisement : Il permet la génération des chromosomes enfants en combinant les caractéristiques (les gènes) des chromosomes parents sélectionnés. Cet opérateur respecte aussi le principe de la théorie de Darwin puisqu'il garantit la survie des chromosomes les plus adaptés. Parmi les opérateurs classiques de croisement, on peut citer les suivants. *Le croisement à un point* qui consiste à sélectionner un point de coupure, puis à subdiviser le génotype de chacun des parents en deux parties. Les fragments obtenus sont alors échangés pour créer le génotype des chromosomes enfants. *Le croisement à deux points* qui est une généralisation du croisement à un point sauf qu'au lieu de choisir un seul point de coupure, on sélectionne deux points de coupure. Finalement, *le croisement*

uniforme qui consiste à générer au hasard un chromosome, dit « chromosome template », pour chaque pair de parents. Si le $i^{ème}$ gène de ce chromosome vaut 1 alors l'échange des allèles dans le $i^{ème}$ locus des deux parents est activé, alors que la valeur 0 pour le $i^{ème}$ gène signifie que l'échange n'est pas autorisé. A noter que l'opérateur de croisement est appliqué aux chromosomes parents avec une probabilité oscillant, généralement, entre 0,50 et 0,95.

- L'opérateur de mutation : Il consiste à modifier quelques gènes des chromosomes enfants. Cet opérateur respecte le principe de variation de la théorie de Darwin. Il est appliqué pour avoir des individus plus adaptés à l'environnement. Un opérateur très simple et fréquemment utilisé pour les chromosomes ayant une représentation binaire est le « *bit flip mutation* » qui consiste altérer la valeur d'un gène de 1 à 0 et vice versa. Cependant, l'opérateur de mutation est appliqué à quelques chromosomes enfants par une faible probabilité oscillant, généralement, entre 0,001 et 0,01.

Ces opérateurs susmentionnés permettent d'évoluer les chromosomes à travers les générations jusqu'à ce qu'un critère d'arrêt soit vérifié. Ce dernier peut correspondre soit à un nombre maximal de générations ou un temps d'exécution fixé a priori ou encore lorsque la solution de l'algorithme atteint une valeur prédéfinie.

En outre, malgré l'efficacité et la simplicité de la mise en œuvre des AGs, ces derniers souffrent du problème de convergence prématurée. C'est pour cette raison que plusieurs travaux de recherche ont été menés pour améliorer la performance des AGs. En effet, l'opérateur de sélection et de croisement jouent un rôle vital dans la performance de ces algorithmes. La sélection des chromosomes parents est une étape cruciale car si les parents sont sélectionnés d'une manière intelligente, elle augmentera la diversité de la population évitant ainsi la convergence prématurée. Cependant, l'utilisation d'un opérateur de croisement approprié est très importante car il permet d'exploiter davantage l'espace des solutions. Ce qui permet ainsi aux AGs d'avoir un équilibre entre l'intensification à travers l'opération de croisement et la diversification grâce à la stratégie de sélection. Dans ce sens, nous avons proposé une version améliorée d'une stratégie de sélection, dite « *sélection sexuelle* », en s'inspirant de la technique introduite par (Varnamkhasti & Lee, 2012a). Nous avons proposé aussi deux variantes d'un opérateur de croisement développé par (Aghezzaf & Naimi, 2009). Ces opérateurs génétiques sont combinés dans un AG, appelée ISGA (pour Improved Sexual Genetic Algorithm). Une telle combinaison a démontré une sorte de synergie et coopération lors de la résolution du problème 0/1 MKP.

En effet, l'opérateur de sélection sexuelle proposé par (Varnamkhasti & Lee, 2012a) prend en considération le fait que la reproduction des chromosomes enfants n'est effectuée que pour des chromosomes ayant un sexe opposé. Pour cela, la sélection sexuelle répartit la population des chromosomes en deux groupes selon leur sexe : Groupe des chromosomes femelles et groupe des chromosomes mâles. Les chromosomes femelles sont sélectionnés par la méthode de

tournois tandis que les chromosomes mâles sont tirés au hasard à partir du groupe des mâles. Par la suite, chaque chromosome femelle choisit son partenaire selon trois critères d'une manière alternative : Distance de Hamming, degré d'adaptation qui est mesuré par la fonction fitness ou le nombre des gènes actifs (c'est à dire les gènes ayant la valeur 1). De notre part, nous avons adapté cette stratégie de sélection à l'espace phénotypique au lieu de l'espace génotypique en appliquant des modifications au niveau des critères de choix des partenaires mâles. On parle de la distance de Manhattan qui remplace la distance de Hamming dans l'espace phénotypique et la taille occupée par chaque objet mis dans le sac au lieu de considérer seulement l'existence ou non d'un objet dans le sac. En d'autres mots, la stratégie de sélection proposée favorise les chromosomes mâles les plus distants dans l'espace phénotypique ; s'il y en a plusieurs, on choisit ceux qui sont les plus adaptés ; s'il y en a plusieurs, on donne la priorité aux chromosomes ayant des objets de petites tailles afin de laisser place à d'autres objets en vue de les ranger dans les prochaines générations. Sinon, un choix aléatoire aura lieu au cas où aucune de ces critères n'est satisfaite. Outre cela, pour bénéficier de la diversification créée par un tel opérateur de sélection et la préserver au cours des générations, nous avons choisi d'incorporer un opérateur de croisement (Aghezzaf & Naimi, 2009) qui apparaît le plus convenable. Cet opérateur produit un seul chromosome enfant au lieu de deux, à partir de deux chromosomes parents en se basant sur deux étapes principales. La première étape favorise l'héritage des gènes similaires, autrement dit, elle garantit le partage des caractéristiques en commun à travers les générations. La deuxième étape améliore la qualité du chromosome en termes de son adaptation, tout en affectant aux gènes non similaires (ayant des valeurs différentes dans le même locus) une fonction fitness spécifique qui donne la priorité aux gènes les plus adaptés. A savoir, cet opérateur a prouvé son efficacité dans la résolution du problème de sac à dos multi-objectif, ce qui nous a encouragé de l'adapter au problème 0/1 MKP en effectuant des changements plus ou moins légers à la deuxième étape de la création du chromosome enfant. Une première modification consiste à attribuer aux gènes une fonction fitness spécifique à la structure du problème. Alors que la deuxième modification injecte l'aspect aléatoire dans le choix des objets à ranger dans le sac.

Les opérateurs proposés sont testés sur une large gamme d'instances issues de la littérature du problème 0/1 MKP. Par la suite, les résultats sont comparés à ceux obtenus par des AGs utilisant les opérateurs de sélection et de croisement définis ci-dessus. Nous avons fondé notre évaluation sur des critères robustes, notamment l'écart en pourcentage, l'erreur moyenne ainsi que des tests statistiques non paramétriques tels que : Le test de Wilcoxon et le test de Friedman qui permettent respectivement de comparer deux à plusieurs échantillons liés. Les expériences expérimentales ont été effectuées en choisissant séparément deux critères d'arrêt qui sont un nombre maximum de générations et un temps fixe. Pour un nombre maximum de générations, on a remarqué que les opérateurs proposés produisent des solutions de bonne qualité par rapport à celles obtenues par les opérateurs classiques, sauf que leur temps consommé est relativement long. En tirant profit du fait que les AGs classiques ont un problème de convergence prématurée, on a donné plus de temps aux opérateurs proposés afin de leur donner plus de

chance d'améliorer leurs résultats tout en fixant en avance un temps de calcul assez suffisant et similaire pour tous les algorithmes sujets à la comparaison. Par conséquent, les résultats obtenus confirment que l'algorithme proposé est toujours plus efficace que les algorithmes comparatifs malgré un temps d'exécution largement suffisant.

Chapitre 4 :

Comme mentionné précédemment, les algorithmes génétiques (AGs) sont des outils d'optimisation performants qui ont prouvé à travers les années leur efficacité sur une gamme diversifiée des problèmes d'ingénierie et d'optimisation. Malgré ceci, comme tout algorithme d'optimisation, les AGs souffrent d'un problème classique qui réside dans la possibilité d'avoir des chromosomes enfants identiques à une génération donnée, ce qui implique une convergence prématurée. Afin de remédier à ce problème, les chercheurs ont procédé à deux techniques différentes : Une modification appliquée à un ou plusieurs opérateurs de l'algorithme, ou encore une hybridation de l'algorithme génétique avec un ou plusieurs autres algorithmes. L'objectif principal de la technique d'hybridation est de combler les lacunes de l'AG tout en tirant profit des avantages du deuxième algorithme d'hybridation. Dans cette thèse, nous choisissons d'incorporer la méthode des *k*-moyennes afin de bénéficier de ses avantages.

Les algorithmes de *clustering* (*classification*) permettent de trouver des groupes d'objets tels que les objets du même groupe sont similaires (distance intra-cluster est minimisée) et les objets des groupes différents sont dissimilaires (distance inter-cluster est maximisée). En effet, on distingue entre deux types de clustering : Le clustering hiérarchique et le clustering en partitionnement. Le premier type permet d'obtenir un ensemble de clusters imbriqués organisés en un arbre hiérarchique tandis que le deuxième type permet de diviser les données à des sous-ensembles qui ne se chevauchent pas. Un exemple facile et simple du clustering en partitionnement est la méthode des *k*-moyennes. Cette méthode construit un cluster de façon qu'un objet appartenant à un cluster est plus proche de son centre que du centre des autres clusters.

Dans le but de créer une diversité génétique au sein de la population et par conséquent remédier au problème de convergence prématurée, nous nous sommes inspirés du principe de la méthode des *k*-moyennes qui permet de diviser la population en *k* partitions ayant des caractéristiques différentes. En effet, nous avons fixé *k* à 2 afin de créer deux groupes distincts. Un groupe sert à choisir le premier parent tandis que l'autre sert à sélectionner le deuxième parent de telle sorte que le couple résultant soit, dans la mesure du possible, plus distant. L'idée d'une telle stratégie de sélection est d'éviter de produire des chromosomes enfants identiques en augmentant le taux de diversité dans la population. L'approche hybride que nous avons proposé dans ce chapitre est dénotée par GA-based *k*-means (Algorithme génétique basé sur la méthode des *k*-moyennes).

GA-based *k*-means commence par générer une population initiale d'une manière aléatoire. Pour ne pas avoir des solutions irréalisables dans la population, il applique un opérateur de réparation et d'amélioration décrit dans l'article de (Zouache, et al., 2016). Cet opérateur repose sur deux étapes. Une étape de réparation qui permet la transformation d'une solution irréalisable en une solution réalisable. En effet, elle trie d'abord les objets selon l'ordre croissant de leurs ratios $p_i / \sum_{j=1}^{d} w_{ij}$, $\forall i = \{1, \dots, N\}$ (avec N est le nombre d'objets et d est le nombre de dimensions du sac, ainsi le numérateur représente le profit de chaque objet alors que le dénominateur correspond à la somme des tailles de ce dernier). Ensuite, elle retire itérativement les objets du sac à dos jusqu'à ce que tous les contraintes de capacité soient satisfaites. La deuxième étape est une étape d'amélioration de la qualité des solutions. Cette fois-ci les objets qui ne sont pas encore rangés dans le sac, sont triés selon l'ordre décroissant de leurs ratios, puis ils sont y réinsérés un par un jusqu'à ce qu'on ne peut ajouter aucun nouvel objet sans violer au moins une contrainte de capacité ou lorsqu'il ne reste plus d'objets à ranger.

Après l'initialisation de la population, notre approche hybride applique la méthode des *k*-moyennes (avec $k = 2$) à la population pour la diviser en deux groupes dissimilaires. Pour que la méthode des *k*-moyennes s'adapte mieux au contexte de l'AG, les chromosomes ont été encodés dans l'espace phénotypique tout en affectant à chaque gène une fonction fitness f. En effet, le i-ème gène reçoit la valeur de la fonction $f(i) = p_i / \sum_{j=1}^{d} w_{ij}$, si l'objet correspondant est placé dans le sac. Sinon, le i-ème gène reçoit la valeur 0. Suite à cette représentation, la méthode des *k*-moyennes applique la distance de Manhattan pour mesurer le degré de similarité entre les chromosomes et affecte ainsi les chromosomes les plus proches au même groupe par le biais d'un centroide (centre du cluster). Dans notre approche, un centroide est un chromosome dont la valeur de chaque gène correspond à la moyenne des valeurs fitness qui apparaissent dans le même locus de tous les chromosomes d'un cluster. L'étape suivante consiste à choisir au hasard un parent du groupe ayant moins de chromosomes. Tandis que pour choisir les partenaires, on applique la stratégie de sélection par tournoi binaire au deuxième groupe de chromosomes. Une fois les couples de parents sont placés dans le « mating pool », on applique un opérateur de croisement pour la reproduction des chromosomes enfants, en particulier le croisement uniforme. Puis, on procède à la mutation de quelques chromosomes enfants en utilisant l'opérateur « *Bit flip mutation* ». Pour la création des nouvelles générations, nous adoptons la stratégie de remplacement générationnel qui remplace la totalité des chromosomes parents par leurs descendants.

Pour tester l'efficacité et la performance de notre approche hybride, nous avons utilisé le groupe d'instances proposé par Glover et Kochenberger en 2000, contenant entre 100 à 2500 objets et entre 15 à 100 contraintes de capacité. La performance de notre algorithme a été mesurée en termes de qualité des solutions moyenne et du temps de calcul moyen. Nous avons ainsi comparé nos résultats avec ceux obtenues par des algorithmes génétiques utilisant des stratégies de sélection traditionnelles ; on parle de la sélection par tournois, la sélection par roulette, la sélection par troncature et la stratégie de sélection stochastique universelle. Les résultats

expérimentaux montrent que le GA-based *k*-means est plus performant que les autres algorithmes de comparaison en termes de qualité des solutions, sauf pour quelques instances dans lesquelles notre approche est dépassée par l'AG adoptant la sélection par roulette. Quant au temps de calcul, notre approche consomme un temps comparable en moyenne avec celui consommé par les approches rivales. Or, pour les instances de grande taille, on remarque une différence un peu plus significative en faveur de ces dernières.

Chapitre 5 :

Le problème du bin packing bidimensionnel (2D-BPP pour two-dimensional bin packing problem) consiste à déterminer le nombre minimum de rectangles identiques (bins) à utiliser pour placer un ensemble donné de rectangles plus petits (objets), sans chevauchement et en respectant la capacité limitée de chaque bin. Les objets sont rangés de telle façon que leurs arêtes soient parallèles à celles des bins qui les contiennent, on parle du rangement orthogonal. Mathématiquement parlant, il existe plusieurs types de modélisations pour le problème 2D-BPP. Malgré cela, il n'existe pas un modèle linéaire efficace comme le cas du problème 0/1 MKP. Nous présentons ainsi deux modèles qui sont très connus dans la littérature du problème 2D-BPP. Le premier modèle a été proposé par (Gilmore & Gomory, 1965) comme une extension de leur approche développée pour le problème du bin packing unidimensionnel (Gilmore & Gomory, 1963). Ce modèle est basé sur l'énumération de tous les sous-ensembles d'objets qui peuvent être rangés dans un même bin sans dépasser sa capacité limitée. Chaque objet doit appartenir à un seul sous-ensemble. La fonction objectif de ce modèle est de minimiser le nombre de configurations de rangements réalisables. Le deuxième modèle est basé sur la théorie des graphes. Il a été proposé par (Fekete & Schepers, 2004) et consiste à tracer deux graphes : un graphe relatif à la hauteur des objets $G_H = (V, E_H)$ et un autre relatif à leur largeur $G_w = (V, E_w)$. Chaque sommet des deux graphes est associé à un objet rangé dans le bin. Les deux graphes sont construits de la façon suivante. Une arête est rajoutée dans G_w (resp. G_H) entre deux sommets si et seulement si les projections sur l'axe horizontal (resp. l'axe vertical) des objets relatifs à ces deux sommets se chevauchent. Fekete et Schepers montrent qu'une solution associée à un couple de graphes d'intervalle (G_H, G_w) est réalisable si les objets ne débordent pas les bins et s'il n'y a pas de chevauchement entre deux objets quelconques rangés dans un même bin.

De nombreux problèmes pratiques se modélisent sous forme du problème 2D-BPP. Cependant, chaque problème réel présente ses propres spécificités en termes des caractéristiques des objets, contraintes du problème, fonction objectif, etc. Ceci a donné lieu à plusieurs variantes du problème 2D-BPP telles que :

- Le problème du strip packing en deux dimensions : C'est une variante très étudiée et fréquemment utilisée dans les heuristiques du problème 2D-BPP. En effet, on dispose d'une

liste d'objets et d'un seul bin de largeur fixe et d'une hauteur infinie. L'objectif est de ranger tous les objets disponibles dans le bin tout en minimisant la hauteur totale à utiliser.

- Le problème 2D-BPP avec possibilité de rotation des objets à 90° : C'est une version non-orientée du problème 2D-BPP, dans laquelle la rotation 90° des objets est permise. Cependant, la rotation des objets n'est pas toujours possible surtout dans les problèmes de découpe (*e.g.* découpe des articles décorés). Cette version est rarement étudiée par les chercheurs.

- Le problème 2D-BPP avec la contrainte guillotine : C'est une variante qui impose des coupes allant de bout en bout des bins pour restituer les objets. La contrainte guillotine est imposée par les caractéristiques des machines automatiques dans les problèmes de découpe. Par contre, cette contrainte n'est plus considérée dans les problèmes de chargement.

- Le problème 2D-BPP avec conflit entre les objets : Un conflit entre deux objets est une contrainte qui interdit de ranger ces deux objets dans le même bin. Cette contrainte peut être jointe, par exemple, aux contraintes des problèmes de chargement des matières inflammables. Il existe un autre type de conflit qui est le conflit partiel. Il s'agit d'une restriction imposant à séparer les objets en conflit partiel par une certaine distance (de sécurité). Nous vous invitons à se référer aux travaux de ((Khanafer, et al., 2012) et (Hamdi-Dhaoui, et al., 2012)) pour plus de détails.

- Le problème du bin packing multidimensionnel : C'est une généralisation du problème 2D-BPP qui considère plus que deux dimensions. Les articles traitant cette variante sont extrêmement rares.

En effet, le problème 2D-BPP se fréquente de façon directe ou indirecte dans de nombreux problèmes industriels. On le retrouve essentiellement dans l'industrie du tissu, de l'acier, du bois ou du verre sous forme de problème de découpe. On le trouve aussi dans d'autres applications comme le problème des tournées de véhicules (Stille, 2008) dans lequel les véhicules correspondent aux bins, tandis que les biens à transporter sont les objets; ou encore dans le problème d'affectation des tâches pour les systèmes robotisés (Stille, 2008) qui vise à trouver un nombre optimal de robots (bins) pour réaliser un ensemble donné des tâches (objets). On le trouve également dans les systèmes de calcul distribué qui consistent à attribuer les outils informatiques de façon à répondre à toutes les requêtes des utilisateurs. Les requêtes correspondent aux objets à ranger tandis que les outils informatiques correspondent aux bins.

Le problème 2D-BPP est une généralisation du problème de bin-packing en une dimension (1D-BPP) qui est connu comme étant un problème NP-difficile, donc il en va de même pour le problème 2D-BPP. Par conséquent, l'énumération de toutes les solutions réalisables pour trouver la meilleure solution s'avère impossible même pour des instances de taille moyenne. Toutefois, on peut recourir à des méthodes approchées telles que les heuristiques et les

métaheuristiques. Les résultats obtenus par ces méthodes peuvent être comparés aux bornes inférieures qui existent dans la littérature du problème. Cependant, nous avons la valeur optimale si la borne supérieure (*i.e.* la solution obtenue par la méthode utilisée) est égale à la borne inférieure. L'objectif principal des bornes inférieures est de juger la qualité des solutions obtenues par les méthodes de résolution approchée. Les bornes inférieures servent aussi comme critères d'arrêt dans les métaheuristiques ou les algorithmes itératifs. De plus, avoir des bornes inférieures de bonne qualité permet aussi de réduire la taille de l'espace des solutions à énumérer quand une méthode exacte est appliquée pour résoudre le problème 2D-BPP. La bibliographie des bornes inférieures du problème 2D-BPP est riche. On trouve la fameuse borne continue qui consiste à sommer la taille des objets d'une instance donnée et la diviser par la taille d'un bin. Cette borne fournit de bons résultats lorsque l'instance est constituée de petits objets. Or, si l'instance présente un pourcentage élevé d'objets de taille moyenne, les résultats obtenus peuvent vite devenir insatisfaisants. (Martello & Vigo, 1998) ont proposé trois bornes inférieures qui consistent à diviser les objets en trois groupes : Groupe des objets de taille petite, groupe des objets de taille moyenne et groupe des objets de taille grande. Leurs bornes inférieures dominent la borne inférieure continue. Récemment, des nouvelles bornes inférieures basées sur le concept des fonctions duales réalisables ont été proposées par (Fekete & Schepers, 2000). Les fonctions duales réalisables sont appliquées aux instances du problème afin de modifier la taille de certains objets et d'améliorer le rapport entre la somme des tailles des objets et la taille d'un bin (*i.e.* la borne continue).

Les méthodes de résolution du problème 2D-BPP, comme tout problème d'optimisation, incluent les méthodes exactes, les heuristiques et les métaheuristiques. Les articles qui portent sur la résolution exacte du problème 2D-BPP sont rarement fréquentés. Ils adoptent en général l'approche de recherche par séparation et évaluation. Par exemple, (Clautiaux, et al., 2007) et (Pisinger & Sigurd, 2007) ont proposé une méthode exacte basée sur l'approche de séparation et évaluation pour le problème 2D-BPP. Cependant, (Martello & Vigo, 1998) ont incorporé leurs bornes inférieures proposées dans un algorithme itérative pour trouver des solutions exactes. Concernant les heuristiques, elles occupent une place importante dans la littérature du bin packing, en particulier le problème 2D-BPP, parce qu'elles consomment un temps de calcul modeste voire linéaire pour trouver des solutions de bonne qualité. Les stratégies utilisées dans le problème 1D-BPP est la source d'inspiration de plusieurs heuristiques consacrées au problème 2D-BPP. Nous citons à titre d'exemples les stratégies suivantes :

- La stratégie Next Fit (NF) : Dans cette stratégie, on ne considère qu'un bin ouvert à la fois. Les objets sont traités selon un ordre donné. Les objets sont rangés successivement dans le bin ouvert tant qu'il y a de place. Si l'objet en cours dépasse la capacité du bin, ce dernier est fermé et un nouveau bin est ouvert. Le Next Fit Decreasing (NFD) consiste à trier les objets par ordre décroissant de leur hauteur et à appliquer la stratégie NF pour les ranger dans les bins disponibles.

- Stratégie First Fit (FF) : Initialement un seul bin est considéré. Les objets sont traités selon un ordre donné. Quand il n'y a plus de place dans le bin ouvert pour ranger l'objet en cours, un nouveau bin est créé mais sans fermer le bin en cours. Le First Fit Decreasing (FFD) consiste à trier les objets par ordre décroissant de leur hauteur et à appliquer la stratégie FF pour les ranger dans les bins disponibles.

- Stratégie Best Fit (BF) : Comme dans la stratégie FF, cette stratégie laisse les bins toujours ouverts. Cependant, le choix du bin dans lequel l'objet en cours sera placé dépend des gaps (espaces vacants) présents dans les bins. Ainsi, l'objet en cours est placé dans le bin ayant le moindre gap parmi les bins qui peuvent le contenir. On parle du Best Fit Decreasing (BFD) quand il s'agit de trier les objets à ranger dans l'ordre décroissant de leur hauteur avant de les ranger dans les bins suivant une stratégie BF.

En effet, les heuristiques du problème 2D-BPP peuvent être divisées en deux familles différentes selon (Lodi, et al., 2002): Les heuristiques en une phase et les heuristiques en deux phases. La première famille consiste à ranger directement les objets dans des bins, tandis que la deuxième famille commence d'abord par ranger les objets dans un bin de hauteur infinie en cherchant une solution pour le problème de strip-packing en deux dimensions (2D-SPP). Dans une deuxième phase, un algorithme de résolution pour le problème 1D-BPP consiste à utiliser la solution résultante du problème 2D-SPP pour construire la solution du problème 2D-BPP. Parmi les algorithmes qui procèdent en une phase, nous trouvons ceux qui placent les objets par niveaux comme l'heuristique Finite First Fit (FFF) et Finite Next Fit (FNF) qui sont développées par (Berkey & Wang, 1987). L'heuristique FFF consiste à trier les objets par ordre décroissant par rapport à leur hauteur. Le premier niveau est défini ainsi par l'objet ayant la plus grande hauteur. Les objets restants sont placés dans le niveau le plus bas du premier bin qui peut les contenir. Si aucun niveau ne peut contenir l'objet en cours, un nouveau niveau est créé dans le bin en cours ou bien un nouveau bin est ouvert tout en initialisant un nouveau niveau mais sans fermer le bin en cours. L'heuristique FNF procède de la même manière que l'heuristique FFF sauf au cas où un objet ne peut pas être rangé dans le bin en cours, un nouveau bin est ouvert alors que le premier est fermé. Parmi les algorithmes qui ne rangent pas les objets par niveau, nous trouvons essentiellement ceux qui adoptent une stratégie connue sous le nom de Bottom-Left. On cite les heuristiques Finite Bottom-Left et Next Bottom-Left proposées par (Berkey & Wang, 1987), qui consistent à ranger les objets dans un ordre donné, en mettant l'objet en cours dans la position la plus en bas et la plus à gauche possible. La seule différence entre les deux stratégies réside dans le fait de fermer ou non le bin en cours lorsqu'il ne peut pas accommoder l'objet en cours. Une autre heuristique a été proposée par (Lodi, et al., 1999), appelée « Alternative Direction », consiste à ranger alternativement les objets de gauche à droite et de droite à gauche. Concernant les méthodes en deux phases, la plupart fonctionnent de la manière suivante: La première étape consiste à trier les objets suivant l'ordre décroissant de leur hauteur et les placer successivement dans un bin de hauteur infinie, en le remplissant niveau par niveau. Chaque niveau est défini par la hauteur du plus long objet accueilli par le

bin et les autres objets sont rangés suivant une stratégie NF, FF, ou BF; on parle respectivement de Next Fit Decreasing Height (NFDH), First Fit Decreasing Height (FFDH), et Best Fit Decreasing Height (BFDH). Aussi parmi les heuristiques en deux phases les plus connues, on cite l'heuristique Hybrid First Fit (HFF) proposée par (Chung, et al., 1982), Finite Best Strip (FBS) proposée par (Berkey & Wang, 1987) et Hybrid Next Fit (HNF) proposée par (Frenk & Galambos, 1987). Les trois heuristiques HFF, FBS, ou HNF consistent premièrement à ranger les objets dans un bin à hauteur infinie selon la stratégie FFDH, BFDH, ou NFDH. Dans une deuxième étape, une solution du problème 2D-BPP est obtenue en appliquant la stratégie FFD, BFD, ou NFD aux niveaux résultants de la première étape (dits blocs). La hauteur de chaque bloc est la hauteur du plus grand objet placé dedans et sa largeur est égale à la largeur du bin. Autres heuristiques en deux phases ont été développées par (Lodi, et al., 1999) incluant l'approche « Knapsack Packing » qui consiste à ranger les objets par niveau en résolvant le problème de sac à dos en une dimension. Ainsi que l'approche « Floor Ceiling » qui range les objets par niveau tout en exploitant l'arête inférieure (ceiling) et l'arête supérieure (floor) du niveau en cours. Si un objet ne tient plus sur le floor d'un niveau, il peut être décalé à droite et accroché au ceiling de ce niveau. La deuxième phase desdites heuristiques consiste à utiliser une méthode exacte pour résoudre le problème 1D-BPP.

Les méthodes approchées consacrées au problème 2D-BPP ne se limitent pas aux heuristiques. Certaines métaheuristiques ont été proposées pour résoudre ce problème. Nous citons la méthode de recherche tabou de (Lodi, et al., 1999), dans laquelle ils ont incorporé leurs heuristiques y compris le « Floor Ceiling » et l'« Alternative Direction » pour guider la recherche de voisinage. (Puchinger & Raidl, 2004) ont proposé une approche hybride qui utilise les algorithmes évolutionnaires au sein d'une méthode exacte (Branch & Price). En sus, (Parreño, et al., 2010) ont proposé une approche qui combine une nouvelle variante de la méthode GRASP (Greedy Randomized Adaptive Search Procedure) basée sur le concept des surfaces maximales avec la méthode VND (Variable Neighborhood Descent). Une surface maximale est définie comme suit: Etant donnée une liste des surfaces vacantes, une surface est dite maximale si elle n'est contenue en totalité dans aucune autre surface. Quelques auteurs ont été intéressés par la résolution du problème 2D-BPP avec les algorithmes génétiques tels que (Gonçalves & Resende, 2013) et (Soke & Bingul, 2006). Tandis que d'autres ont focalisé leurs attentions aux algorithmes d'optimisation par essaim de particules tels que (Liu, et al., 2006), (Liu, et al., 2008) et plus récemment (Shin & Kita, 2017).

Chapitre 6 :

Les méthodes d'optimisation basées sur l'intelligence en essaim deviennent de plus en plus une tendance de recherche très active cette dernière décennie. Elles forment une branche d'algorithmes inspirés des phénomènes naturels tels que l'auto-organisation de certaines espèces, leur comportement collectif et leur coopération pour chercher de la nourriture ou faire face aux prédateurs. Comme le cas des oiseaux, des poissons et des abeilles, qui se regroupent

en essaim dont les membres travaillent et communiquent pour le bien du groupe. L'algorithme de colonies de fourmis (ACO) proposé par (Dorigo, et al., 1996) et l'algorithme d'optimisation par essaim de particules (PSO) développé par (Kennedy & Eberhart, 1995), sont les premiers algorithmes basés sur l'intelligence en essaim. La métaheuristique PSO a été premièrement destinée aux problèmes d'optimisation continue. Deux années après la publication de la version continue, (Kennedy & Eberhart, 1997) ont suggéré une version binaire afin de traiter les problèmes d'optimisation combinatoire. Par contre, dès sa première apparition, la métaheuristique ACO a été conçue de façon qu'elle puisse être adaptée à n'importe quel problème d'optimisation.

Récemment, un algorithme appelé la recherche des corbeaux (CSA) a été introduit par (Askarzadeh, 2016) pour résoudre les problèmes d'optimisation continue. Les études expérimentales menées pour vérifier la performance d'un tel algorithme ont prouvé son efficacité en termes de qualité des solutions et du temps de calcul. L'algorithme CSA est basé sur les principes suivants :

- Les corbeaux vivent en groupes,
- Les corbeaux placent le reste de leur nourriture dans des cachettes: Ils aiment avoir de quoi se nourrir le lendemain,
- Ils sont capables de mémoriser leurs cachettes de nourriture,
- Ils sont connus par leur pillerie: Ils leur arrivent de suivre leurs congénères pour piller leurs nourritures,
- Ils ont la faculté de prévoir les comportements de leurs congénères, ce qui leur permet de protéger leur nourriture contre un futur chapardage.

L'algorithme CSA met en jeu un ensemble de corbeaux pour la résolution d'un problème donné. Cet ensemble est appelé « essaim ». L'essaim des corbeaux survole dans un espace de recherche multidimensionnel. Un corbeau se déplace dans l'espace de recherche soit pour cacher sa nourriture pour un futur repas, soit pour suivre un autre corbeau afin de découvrir sa cachette et piller sa nourriture. Cependant, chaque corbeau a une position dans cet espace à l'itération t et une mémoire qui garde la meilleure position dans laquelle il a caché sa nourriture. Les positions représentent ainsi les solutions potentielles du problème traité. En effet, l'algorithme CSA procède de la manière suivante :

- La première étape de cet algorithme, comme n'importe quel algorithme, commence par l'initialisation des paramètres nécessaires pour le bon déroulement du processus de recherche de la meilleure solution. Ces paramètres sont: La taille de l'essaim N, le nombre maximum d'itérations, la probabilité AP, et la longueur du vol des corbeaux fl.
- Puis il positionne aléatoirement N corbeaux dans un espace multidimensionnel et initialise la mémoire de chaque corbeau par sa position initiale.
- Ensuite, il vérifie la réalisabilité des positions et calcule leurs valeurs fitness.

- Puis, il met à jour les positions des corbeaux par le biais d'un autre corbeau choisi aléatoirement dans l'essaim: Chaque corbeau i choisit un corbeau j pour le suivre afin de découvrir sa cachette et piller sa nourriture. En fait, le déplacement des corbeaux est influencé par deux paramètres: La longueur du vol (fl) et la probabilité (AP) pour qu'un corbeau soit conscient de l'intention du corbeau pilleur. Si cette probabilité est faible, le corbeau i met à jour sa position grâce à sa position précédente, la longueur du vol et la mémoire du corbeau j. Sinon, le corbeau j rebrousse chemin pour orienter son rival vers une mauvaise piste et par conséquent le corbeau i se déplace vers une position aléatoire dans l'espace.
- Après la création des nouvelles solutions, l'algorithme passe à les évaluer par une fonction fitness pour mettre à jour les mémoires des corbeaux dans l'étape suivante.
- L'algorithme entame l'ensemble des itérations permettant la génération de nouvelles solutions dans le but d'améliorer la qualité de la meilleure solution rencontrée par l'essaim, jusqu'à un nombre maximal d'itérations.
- Finalement, l'algorithme retourne la meilleure mémoire de tout l'essaim comme une solution finale du problème traité.

Malgré le jeune âge de l'algorithme CSA, sa simplicité, le nombre de paramètres qu'il utilise et sa performance ont suscité l'intérêt de plusieurs chercheurs. Il a résolu efficacement un nombre important de problèmes d'optimisation tels que les problèmes d'ordonnancement (Adhi, et al., 2018), le problème d'optimisation d'énergie dans les systèmes électriques (Díaz, et al., 2018) et quelques problèmes d'optimisation binaire comptés sur les bouts des doigts. Dans notre thèse, nous avons utilisé l'algorithme CSA pour résoudre le problème 2D-BPP. A notre connaissance, c'est la première application de l'algorithme CSA aux problèmes de bin packing en général, et au problème 2D-BPP en particulier. En effet, nous avons proposé une version binaire de cet algorithme afin qu'il s'adapte à l'espace de recherche binaire du problème 2D-BPP. Il est dénoté par l'acronyme BCSA pour Binary Crow Search Agorithm. Dans le processus de binarization de l'algorithme CSA, nous avons utilisé une technique dite « transformation Sigmoid ». Cette dernière a été appliquée la première fois à l'algorithme PSO (Kennedy & Eberhart, 1997) pour le binariser. Plus récemment, elle a été appliquée à l'algorithme des algues artificielles (Zhang, et al., 2016) et l'algorithme de pollinisation des fleurs (Abdel-Basset, et al., 2018). Cette technique nous indique la probabilité qu'une variable de décision prenne la valeur 1 ou 0, et par la suite c'est grâce à un nombre généré aléatoirement comparé à cette probabilité qu'on peut binariser les valeurs des variables de décision.

Dans notre algorithme BCSA, nous avons encodé les positions sous forme de matrices binaires dont le nombre de lignes correspond au nombre d'objets à ranger et le nombre de colonnes correspond au nombre de bins disponibles. Chaque coefficient de cette matrice représente la valeur d'une variable de décision qui indique si un objet et rangé dans un bin donné ou non. Un bin est considéré comme un bin utilisé si la somme des coefficients appartenant à la même colonne est supérieure ou égale à 1. Dans la première étape, les positions sont initialisées par

des 0 ou 1 d'une manière aléatoire à condition qu'un objet sera placé une et une seule fois dans un bin donné. Autrement dit, la somme des coefficients de la même ligne est égale à 1. On note que la stratégie adoptée pour le placement des objets est la stratégie Bottom Left. La qualité des positions est évaluée grâce à la fonction fitness proposée par (Falkenauer & Delchambre, 1992). Cette fonction consiste à maximiser le contenu des bins sans dépasser leur capacité limitée. A l'étape qui suit, les positions sont mises à jour par la façon suivante : Si la probabilité (AP) est faible on met à jour les valeurs de la matrice en appliquant la transformation Sigmoid à une fonction qui dépend de la longueur du vol, la mémoire du corbeau à suivre et la position actuelle du corbeau en cours. Sinon, les positions des corbeaux sont mises à jour aléatoirement. Les positions générées sont ainsi évaluées, ce qui sert après à mettre à jour les mémoires des corbeaux. Ces étapes sont répétées jusqu'à atteindre un nombre maximum d'itérations qu'on l'a fixé à 100. Finalement, la meilleure mémoire est reportée pour désigner la solution finale du problème 2D-BPP.

Pour évaluer la performance de notre algorithme binaire, on l'a testé sur un ensemble de d'instances du problème 2D-BPP. Cet ensemble regroupe les instances en dix classes. Six classes sont proposées par (Berkey & Wang, 1987) et quatre classes sont proposées par (Martello & Vigo, 1998). Chaque classe est divisée en cinq sous classes selon le nombre des objets à ranger, on trouve 20, 40, 60, 80 et 100 objets. Pour chaque sous classe on trouve dix instances ayant différentes tailles d'objets et diverses complexités. Cependant, chaque classe est dotée d'une même capacité des bins. Dans notre simulation, nous avons choisi de chaque sous classe une seule instance d'une manière aléatoire, jouant ainsi le rôle d'un individu représentatif d'un échantillon. Les résultats expérimentaux sont comparés à l'heuristique Finite First Fit (FFF) de (Berkey & Wang, 1987) et l'algorithme PSO binaire de (Kennedy & Eberhart, 1997). L'algorithme PSO binaire et le BCSA ont été exécutés 30 fois indépendamment puisqu'ils ont une nature stochastique. On a remarqué ainsi d'après les résultats obtenus que notre algorithme surpasse en moyenne l'algorithme PSO binaire et l'heuristique FFF en termes de qualité des solutions. Concernant le temps de calcul, le BCSA converge rapidement par rapport à l'algorithme PSO binaire. Tandis que les deux algorithmes consomment un temps couteux par rapport à l'heuristique FFF et ceci est tout à fait logique puisque cette dernière ne nécessite que quelques opérations à effectuer.

Dans des travaux futurs, l'algorithme BCSA peut être utilisé pour résoudre les différentes variantes du problème de bin packing. Il peut être aussi utilisé pour résoudre autres problèmes d'optimisation combinatoire en appliquant des modifications mineures.

Chapitre 7 :

Dans ce chapitre, nous avons proposé une autre technique qui est simple et fructueuse pour adapter l'algorithme CSA au contexte binaire du problème 2D-BPP. Cette technique est l'hybridation de l'algorithme CSA avec un autre algorithme binaire tel que l'algorithme de

colonies de fourmis, l'algorithme génétique, ou encore l'algorithme PSO binaire. Dans cette thèse, nous avons opté pour l'algorithme génétique pour trois raisons :

- Il peut être appliqué à n'importe quel type d'optimisation (continue, discrète, ou binaire).
- Il a prouvé son efficacité dans la résolution des problèmes de chargement, en particulier le problème 2D-BPP. Par conséquent, il pourra améliorer la performance de l'algorithme CSA tout en assurant un équilibre entre l'exploration de l'espace de recherche et l'exploitation de ces zones prometteuses.
- Les opérateurs génétiques peuvent être facilement incorporés au sein de l'algorithme CSA, surtout que ce dernier se base sur des étapes à peu près similaires à celles de l'algorithme génétique. On cite la sélection des corbeaux à suivre qui correspond à la sélection des chromosomes parents, ainsi que la mise à jour des positions des corbeaux qui correspond à la reproduction des chromosomes enfant à travers le croisement des chromosomes parents.

Cependant, il existe quelques chercheurs qui ont envisagé la combinaison de l'algorithme CSA avec d'autres méthodes de résolution, afin de tirer profit des avantages de chaque méthode et de combler leurs lacunes. A titre d'exemple, on cite l'approche développée par (Allaoui, et al., 2018), qui combine la méthode de recherche locale avec l'algorithme CSA. Leur méthode hybride a été appliquée au problème d'assemblage de fragments d'ADN. (Lakshmi, et al., 2018) ont incorporé l'algorithme CSA dans la méthode des *k*-moyennes afin d'échapper des optima locaux. (Pasandideh & Khalilpourazari, 2018) ont hybridé l'algorithme CSA avec l'algorithme Sine Cosine (Mirjalili, 2016). Ainsi, l'algorithme résultant a été testé sur un benchmark de fonctions continues. Plus récemment, (Arora, et al., 2019) ont développé une approche hybride pour résoudre le problème de sélection des caractéristiques (Feature selection problem), qui combine l'algorithme CSA avec l'algorithme des loups gris (Mirjalili, et al., 2014).

Quant à notre approche hybride, nous avons incorporé les opérateurs de l'algorithme génétique dans le processus de l'algorithme CSA, à savoir : l'opérateur de sélection, l'opérateur de croisement et l'opérateur de mutation. L'algorithme résultant, nommé CSGA (pour Crow Search-based Genetic Algorithm), se caractérise par un bon équilibre entre l'exploitation et l'exploration de l'espace de recherche et peut résoudre tous types de problèmes d'optimisation (continue, discrète, et binaire) à l'opposé de l'algorithme BCSA et CSA. En effet, les positions dans l'algorithme CSGA se considèrent comme des chromosomes de l'algorithme génétique, aussi bien que les mémoires des corbeaux qui sont les chromosomes les plus adaptés à leur environnement. Les positions (ou chromosomes) sont encodées par des matrices binaires comme le cas des positions dans l'algorithme BCSA. La réparation des solutions irréalisables se fait par l'opérateur de réparation qu'on a proposé dans cette thèse et qui s'applique directement aux bins dont la capacité est violée. L'évaluation de la qualité des positions se fait grâce à une fonction fitness égale à l'inverse de la fonction objective. En d'autres mots, plus la fonction fitness est maximale plus le nombre de bins utilisés est minimal. La sélection des corbeaux à suivre par leurs congénères pour découvrir leurs cachettes s'effectue aléatoirement dans l'algorithme d'origine CSA. Alors que dans l'algorithme CSGA, on applique une stratégie

de sélection, plus précisément la sélection par tournois. La mise à jour des positions se fait, dans l'algorithme CSA, grâce à une équation qui dépend de la position en cours, de la mémoire du corbeau suivi ainsi que des paramètres AP et fl. Tandis que dans notre algorithme CSGA, on applique un opérateur de croisement comme suit : Si la probabilité AP est faible, la position actuelle du corbeau est croisée avec la position actuelle du corbeau suivi. Sinon, la position actuelle du corbeau est croisée avec sa mémoire (sa meilleure position). De plus, nous avons ajouté un autre opérateur de mutation dans le but de créer une diversification dans l'essaim. Brièvement, les étapes de l'algorithme CSGA sont décrites comme suit :

- L'algorithme commence par l'initialisation des paramètres nécessaires: La taille de l'essaim, le nombre maximum de générations, la probabilité AP, et la probabilité de mutation.
- Puis, il positionne aléatoirement les corbeaux dans l'espace multidimensionnel et initialise leur mémoire par leur position initiale.
- Ensuite, il vérifie la réalisabilité des solutions et applique l'opérateur de réparation aux solutions irréalisables, puis il calcule leur valeur fitness.
- Puis, chaque corbeau i choisit un corbeau j pour le suivre grâce à la fameuse méthode « sélection par tournoi binaire ».
- Par la suite, l'algorithme met à jour la position de chaque corbeau en croisant sa position actuelle par sa mémoire ou bien sa position actuelle par la position du corbeau sélectionné. Cela dépend d'un seul paramètre qui est la probabilité AP qu'on peut le définir tout simplement par le degré de prudence chez les corbeaux. On note que l'opérateur de croisement utilisé dans notre algorithme CSGA est l'opérateur de croisement PMX proposé par (Goldberg & Lingle, 1985).
- Après l'opération de croisement, l'algorithme applique un opérateur de mutation inspiré de celui introduit par (Liu, et al., 2006) sur quelques positions générées selon une probabilité très faible (entre 0.001 et 0.01). L'opérateur de mutation consiste à choisir au hasard un bin utilisé et le diviser en deux. La première partie des objets est conservée dans son bin d'origine et la seconde partie est fusionnée dans un autre bin qui peut l'accommoder sans violer sa contrainte de capacité. Sinon, si aucun bin ne peut l'accueillir un nouveau bin est ouvert.
- Puis, l'algorithme calcule la qualité de la position de chaque corbeau et met à jour sa mémoire par le biais d'une fonction fitness.
- Les opérations décrites ci-dessus se répètent jusqu'à la satisfaction du critère d'arrêt.
- Finalement, l'algorithme retourne la meilleure mémoire trouvée dans tout l'essaim en la considérant comme solution finale du problème.

Pour tester l'efficacité et la performance de notre algorithme CSGA, nous l'avons appliqué à quelques instances de (Berkey & Wang, 1987) et (Martello & Vigo, 1998). On n'a considéré quatre classes dans nos simulations. Trois classes, qui semblent représentatives, sont choisies parmi celles de (Berkey & Wang, 1987). A côté de ces trois classes, nous avons sélectionné une

quatrième classe parmi celles de (Martello & Vigo, 1998) qui est aussi considérée comme représentative de ses congénères. Cependant, dans chaque classe sélectionnée, nous n'avons considéré que trois sous-classes qui sont variées et caractérisées par différentes complexités: Une sous-classe ayant 20 objets pour représenter des instances de petite taille, les deux autres aient respectivement 80 et 100 objets, représentant respectivement des instances de taille moyenne et large. Les résultats expérimentaux de l'algorithme CSGA ont été comparés en moyenne avec ceux obtenus par des algorithmes de même nature, à savoir l'algorithme PSO binaire proposé par (Kennedy & Eberhart, 1997) et l'algorithme génétique standard. Une première comparaison a été menée grâce au test statistique de Wilcoxon qui vise à vérifier est ce qu'il y a une différence significative entre deux algorithmes ou non en termes de qualité des solutions. Une deuxième comparaison a été effectuée en matière de qualité des solutions et de rapidité de recherche. Les résultats des simulations appliquées sur les instances de (Berkey & Wang, 1987) ont démontré que l'algorithme hybride proposé est meilleur que l'algorithme génétique en termes de qualité des solutions. Cependant, il est parfois équivalent à l'algorithme PSO binaire et parfois surpassé par ce dernier. Par contre, pour la classe représentative de (Martello & Vigo, 1998), notre algorithme produit des meilleures solutions en un temps de calcul moins coûteux par rapport à l'algorithme PSO et l'algorithme génétique.

Conclusion et perspectives

Nous avons traité dans cette thèse le problème de sac à dos multidimensionnel binaire et le problème du bin-packing bidimensionnel. L'importance de ces deux problèmes prouvés comme étant NP-difficiles, réside dans leur utilisation vaste dans des domaines divers tels que le transport et la logistique, l'industrie, les systèmes de télécommunications et réseaux…. Notre apport est montré dans la proposition de deux variantes d'un algorithme génétique pour la résolution du problème de sac à dos multidimensionnel binaire, ainsi que deux versions binaires de l'algorithme de recherche des corbeaux pour la résolution du problème du bin packing bidimensionnel. Ces méthodes proposées ont été testées expérimentalement sur des jeux d'essai existants dans la littérature des deux problèmes. Les résultats obtenus démontrent clairement leur efficacité en termes de qualité des solutions et du temps de calcul.

Comme perspectives, on peut noter d'abord que les algorithmes proposés dans cette thèse sont des algorithmes prometteurs, ce qui encourage de les appliquer pour la résolution d'autres problèmes d'optimisation binaire issus de différents domaines. Autres perspectives, c'est d'améliorer la performance des algorithmes proposés pour le problème du bin packing bidimensionnel en appliquant des nouvelles stratégies de recherche et/ ou en adoptant des nouvelles techniques de binarisation. Ou encore, améliorer la performance des algorithmes génétiques proposés pour le problème de sac à dos multidimensionnel tout en utilisant des nouvelles stratégies inspirées de celles proposées dans cette thèse, ou en intégrant des mécanismes de parallélisme dans le processus de recherche afin d'accélérer le temps de convergence.

Contents

List of figures

List of tables

List of algorithms

Introduction

The packing problems consider a given set of articles (items) having fixed or variable dimensions, and one or more container(s) (also called bag(s), bin(s), or strip(s)…). The main goal is allocating items to the container(s) by taking into account the constraints imposed by the problem at hand. The packing problems arise in many fields: A trivial example is logistics (aircraft containers loading, shipping vessels loading,…), also the industry (cutting materials, tasks scheduling,…), telecommunications (spectrum allocation,…), informatics (grid computing, material resource allocation,…), to mention just a few. The enormous practical applications of the packing problems have given rise to several classifications. A first typology was proposed by (Dyckhoff, 1990) and then extended to a general one by (Wäscher, et al., 2007), which they have called cutting and packing (C&P) problems. The basic types of C&P problems have been developed by combining two criteria "type of assignment" and "assortment of items". The type of assignment indicates the main goal of the problem (*i.e.* maximization or minimization problem). The maximization problem has limited quantities of containers that do not allow for accommodating all available items. So, the objective of such problem is maximizing the profit of used containers. In contrast, in the minimization problem, the number of containers is large enough to accommodate all available items. Consequently, the number of containers has to be minimized. In this thesis, we consider two problems arising from two different kinds of assignment (see Figure 2): The knapsack problems (maximization problem) and the bin packing problems (minimization problem). For the first type problem, we are interested in the binary variant of knapsack problems in which the knapsack has multiple dimensions, which is "0/1 multidimensional knapsack problem (0/1 MKP)". In the second one, we focused on a variant of the bin packing problem that takes into account two dimensions that are the height and width, named "two-dimensional bin packing problem (2D-BPP)".

The two considered packing problems have financial stakes. In other words, an efficient packing reduces the costs, thereby maximizing earnings. That is why, in addition to the research community, they also concern the decision-makers in companies.

The 0/1 MKP and the 2D-BPP are considered as classical problems of combinatorial optimization that consists of finding an optimal solution from a finite set of feasible solutions. Many combinatorial optimization problems, in spite of their easy statement, are difficult to solve, mostly when the problem size is quite large (in this case, the exhaustive search in solutions space will be intractable). A prime example is that of our considered packing problems that belong to the class of NP-hard problems (Martello & Toth, 1990) in which the problems cannot be solved in polynomial time, especially for large-scaled instances.

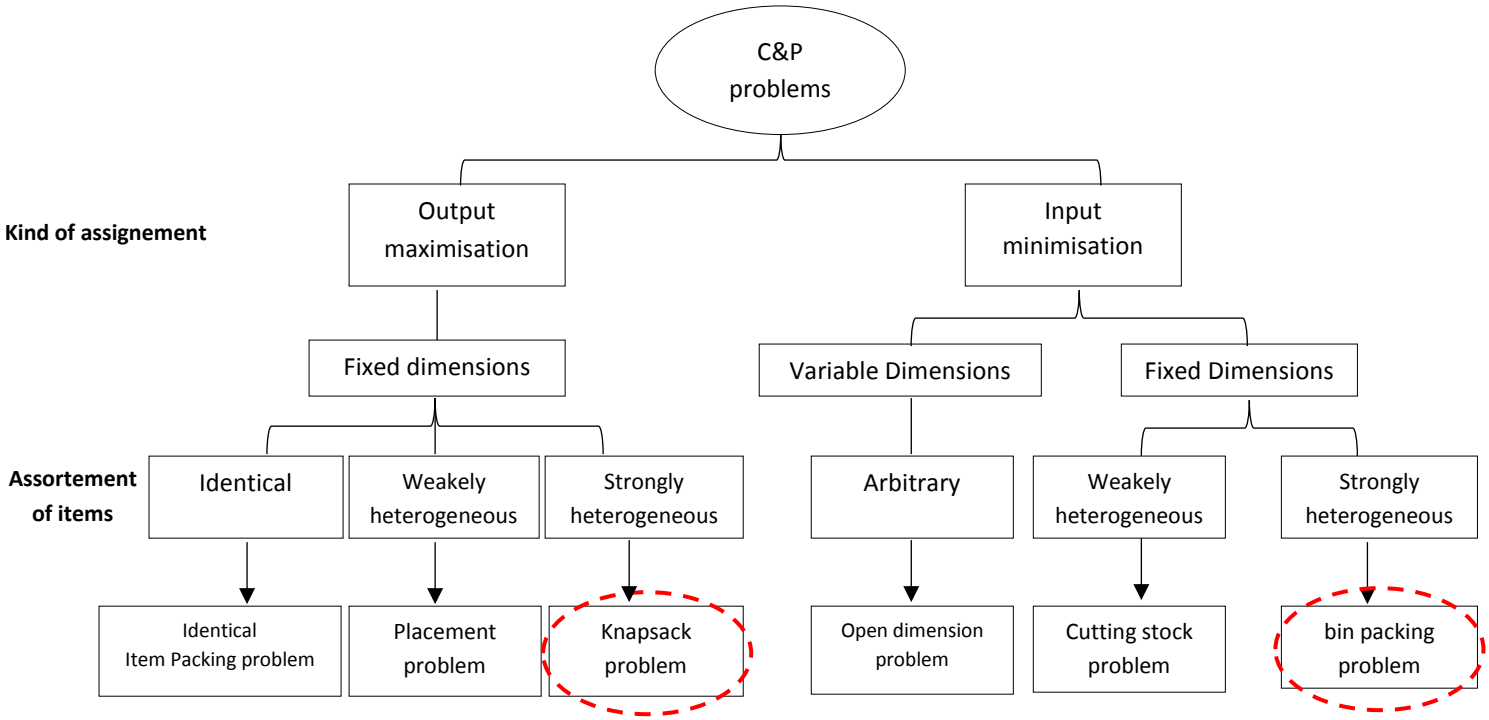


Figure 2. The basic types of C&P problems

There are two categories of methods to settle NP-hard problems: Exact methods and heuristic approaches. The exact methods enumerate all solutions in order to find the optimal one. These methods are efficient only for small to moderate sized problems. However, when the problem size increases, the computation time required to solve it increases exponentially or worse. Heuristic approaches, on the other hand, aim to find a good solution in a shorter execution time without ensuring the optimality of solutions. Heuristic approaches are divided into two classes: Heuristics and metaheuristics. A heuristic is designed to a specific optimization problem, on the contrary, a metaheuristic is an algorithm that can be generalized to many optimization problems by applying minor modifications. Metaheuristics consist of non-specific knowledge problem strategies (*i.e.* they are independent of the problem structure) that control the search process in solutions space. Metaheuristics must respect two major factors: Diversification and intensification (Blum & Roli, 2003). Diversification means exploring efficiently the solutions search space hoping that new regions will be found, while intensification means focusing the search in some local regions that seem to be promising (*i.e.* that can likely include high-quality solutions). A good balance of these two major factors will usually ensure that the optimality is likely achievable.

A wide large number of metaheuristics ranging from very simple schemas which implement basic search processes such as descent method (Cauchy, 1847) or the local search method; to much more complex and meticulous schemas that are in most cases, inspired from successful processes in nature including biological systems, physical and chemical processes. We cite as examples the simulated annealing method that is inspired from a metallurgical process (Kirkpatrick, et al., 1983), genetic algorithms which are inspired from the Darwinian evolution process (Holland, 1975), flower pollination algorithm that imitates the physiological process of

mating in plants (Yang, 2012). We can also cite swarm intelligence algorithms that are inspired by the self-organization and the collective behavior of some social species, such as particle swarm optimization algorithm (Eberhart & Kennedy, 1995), ant colony optimization algorithm (Dorigo, et al., 1996), bee colony algorithm (Karaboga, 2005), and crow search algorithm (Askarzadeh, 2016).

Contributions

Our main objective is solving the 0/1 MKP and 2D-BPP by means of new heuristic approaches, precisely by metaheuristics. Experimental results of the proposed metaheuristics have been carried out on standard instances existing in the literature of the two problems and compared to those reached by analogous nature algorithms. The contributions of this thesis are the following:

First contribution: We present an improved sexual genetic algorithm to solve the 0/1 MKP (Laabadi, et al., 2019a). Firstly, we develop a new variant of a sexual selection strategy introduced by (Varnamkhasti & Lee, 2012a). In fact, Varnamkhasti and Lee's selection strategy takes into account the gender of chromosomes in building couples of parent chromosomes. It consists of dividing the initial population into a group of female chromosomes and a group of male chromosomes. The female chromosomes that will participate in reproduction, are chosen by applying a classical selection method, called "Tournament selection". However, the mate of each selected female chromosome is chosen from the group of male chromosomes by using some preference criteria: Hamming distance between a female chromosome and male chromosomes, fitness function, or active genes of male chromosomes. Knowing that the Varnamkhasti and Lee's selection strategy works in genotype space, our contribution consists of adapting this strategy in phenotype space in order to enhance the quality of solutions by taking advantage of the structure of 0/1 MKP. Secondly, we propose two versions of a crossover operator introduced by (Aghezzaf & Naimi, 2009), which they called "two-stage recombination operator". This operator aims at preserving the similar genes of the chromosome parents. It is based on two steps: In the first step, the offspring chromosome inherits the similar genes of their parent chromosomes. In the second step, the offspring chromosome inherits the non-similar genes by using a preference function. The two-stage recombination operator was firstly developed for the multi-objective knapsack problem. In our thesis, we adapt it to the context of 0/1 MKP by using a convenient preference function. In addition to this first proposed version of the two-stage recombination operator, we present a second one by injecting randomness in its second stage in order to increase the genetic diversity. Finally, we combined the proposed sexual selection strategy with one of the two versions of the two-stage recombination operator. According to simulation experiments, these two synergetic operators have enabled a thorough exploration of solutions search space and consequently they have avoided the premature convergence problem of genetic algorithms.

Second contribution: We propose a hybrid approach by incorporating a well-known clustering method, namely *k*-means method, into a genetic algorithm (Laabadi, et al., 2018c). This approach is devoted to solving 0/1 MKP. Basically, the *k*-means method consists in grouping observations (numerical data, textual data,…) in clusters so that the individuals in the same cluster are as similar as possible and the individuals belonging to different clusters are distant as much as possible. Thus, inspiring by the principle of this latter, we build easily the couples of parent chromosomes by fixing the parameter *k* to 2. Firstly, we divide the initial population into two groups of chromosomes using *k*-means method. Secondly, we built the couples of parents by picking out one chromosome from one group and its mate from the other one. Indeed, all individuals of the group having a lower cardinal will participate in mating pool. Each individual of this group selects his mate from the second group by making a competition between them thanks to the tournament selection method. It is clear that *k*-means method enhances the genetic diversity in the population since it chooses parents having distant characteristics. On the other hand, using the tournament selection method that selects chromosomes according to their performance, allows exploiting efficiently the solution search space. Consequently, the selection strategy based on *k*-means method makes an equilibrium between diversification and intensification at an early stage of the genetic algorithm.

Third contribution: For tackling 2D-BPP, we are interested in swarm intelligence algorithms, particularly the crow search algorithm. This recent metaheuristic was introduced by (Askarzadeh, 2016). The author was inspired by the intelligence of crows to hide their foods while protecting them to be pilfered, as well as their ability to memorize the best hiding place. The crow search algorithm was originally developed to settle continuous optimization problems. In our thesis, we binarize this metaheuristic by using a binarization technique, namely the "Sigmoid transformation", so that it is adapted to binary search space of 2D-BPP (Laabadi, et al., 2020).

Fourth contribution: We propose a second technique to extend the crow search algorithm to binary optimization problems (Laabadi, et al., 2019b). This technique consists of hybridizing the crow search algorithm with a genetic algorithm. The idea is to update solutions (position of crows) by using genetic operators (selection, crossover, and mutation) while preserving the main steps of the crow search algorithm. Such combination aims to binarize the crow search algorithm, but also to balance between diversification and intensification when exploring the solution search space. An additional advantage of this combination is that it can be applied to any type of optimization problem (binary, discrete, or continuous optimization problem).

Thesis overview

In addition to this introduction and an upcoming conclusion, this dissertation is divided into two independent parts: The first part is devoted to the 0/1 multidimensional knapsack problem, whilst the second one concerns the two-dimensional bin packing problem.

Part I: The 0/1 multidimensional knapsack problem

This part is organized as follows:

- *Chapter 1. Definition, variants, and real-world applications*: This chapter introduces some concepts of optimization theory, defines the 0/1 MKP, and provides background about its extensions and variants. This chapter also discusses the practical applications of 0/1 MKP and their mathematical formulations.

- *Chapter 2. Heuristic approaches*: This chapter discusses some exact methods commonly used to solve 0/1 MKP, and provides a recent survey of heuristic and metaheuristic approaches developed this last decade for tackling the problem at hand.

- *Chapter 3. An improved sexual genetic algorithm for solving 0/1 MKP*: This chapter describes the proposed sexual selection strategy as well as two new versions of the two-stage recombination operator. Then, it presents the overall genetic algorithm that combines these proposed operators.

- *Chapter 4. A genetic algorithm based on k-means method for solving 0/1 MKP*: This chapter provides a background of hybrid approaches existing in the literature of 0/1 MKP. Then, it presents the proposed hybrid approach for solving 0/1 MKP that incorporates the *k*-means method in a genetic algorithm.

Part II: The two-dimensional bin packing problem

This part includes three chapters:

- *Chapter 5. Definitions, variants, and resolution approaches*: This chapter introduces the 2D-BPP and discusses various restrictions of the problem. It presents relevant variants of 2D-BPP and cites some real-world applications. Then, it provides background about lower bounds, exact methods, heuristic, and metaheuristic approaches existing in the literature of the considered problem.

- *Chapter 6. A binary crow search algorithm for solving 2D-BPP:* This chapter shows the applications of the crow search algorithm to solve miscellaneous optimization problems. Then, it explains the proposed binary version of the crow search algorithm to solve 2D-BPP.

- *Chapter 7. A crow search based genetic algorithm for solving 2D-BPP:* This chapter gives a background of hybrid approaches existing in the literature of 2D-BPP. Then, it presents the proposed hybrid approach combining the crow search algorithm with a genetic algorithm to resolve the current problem.

Publications & communications:

Publications :

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2020). A Binary Crow Search Algorithm for Solving Two-dimensional Bin Packing Problem with Fixed Orientation. Presented in the *International Conference on Computational Intelligence and Data Science (ICCIDS 2019)*, Gurugram, India, September 6-7, 2019. *Procedia Computer Science*, Vol. 167, pp. 809-818. DOI: https://doi.org/10.1016/j.procs.2020.03.420.

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2019). A Crow Search-Based Genetic Algorithm for Solving Two-Dimensional Bin Packing Problem. Presented in the 42nd *German Conference on Artificial Intelligence (KI 2019)*, Kassel, Germany, September 23-26, 2019. *Advances in Artificial Intelligence*; part of *Lecture Notes in Computer Science*, Vol. 11793, pp. 203-215. Springer. DOI: https://doi.org/10.1007/978-3-030-30179-8_17.

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2019). An improved sexual genetic algorithm for solving 0/1 multidimensional knapsack problem. *Engineering Computations*, Vol. 36 No. 7, pp. 2260-2292. DOI: https://doi.org/10.1108/EC-01-2019-0021.

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2018). The 0/1 Multidimensional Knapsack Problem and Its Variants: A Survey of Practical Models and Heuristic Approaches. *American Journal of Operations Research*, Vol. 8, pp. 395-439. DOI: https://doi.org/10.4236/ajor.2018.85023.

Communications in international conferences

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2018). A genetic algorithm based on *k*-means method for solving 0/1 multidimensional knapsack problem. *Presented in the International Conference of Computer Science and Renewable Energies (ICCSRE'2018)*. Ouarzazate, Morocco, November 22-24.

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2018). A hybrid genetic algorithm for solving 0/1 Knapsack Problem. *Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications (LOPAL'2018)*, Article No. 51, 6 pages. Rabat, Morocco, 2-5 May. ACM. DOI: https://doi.org/10.1145/3230905.3230907.

- S. Laabadi (2016). A new algorithm for the Bin-packing problem with fragile objects. *Proceedings* of *the 3rd International Conference on Logistics Operations Management (GOL'2016)*, pp. 1-7. Fez, Morocco, 23-25 May. IEEE. DOI: https://doi.org/10.1109/GOL.2016.7731722

- S. Laabadi (2015). Bin packing problem or how to pack the objects into the smallest possible number of boxes. *In the 5th Edition of the International Conference on Intelligent Textiles & Mass Customization (ITMC'2015).* Casablanca, Morocco, 3-6 November, 2015.

Communications in national scientific manifestations

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2019). A binary particle swarm optimization algorithm for solving 2D bin packing problem. *Actes de la Septième Journée des Sciences de l'Ingénieur (JSI 2019)*. Casablanca, Maroc, 22 Juin, 2019.

- S. Laabadi, M. Naimi, H. El Amri & B. Achchab (2017). Le problème de sac à dos multidimensionnel: Variantes binaires, applications réelles et approches heuristiques. *Actes de la Cinquième Journée des Sciences de l'Ingénieur (JSI 2017)*. Casablanca, Maroc, 22 Avril, 2017.

Part I: The 0/1 multidimensional knapsack problem

Chapter 1: Definition, variants, and real-world applications

1 Introduction

Every aspect of human life is determined by the result of decisions. A decision process must be quantitatively formulated. In other words, the input data of a decision process must be measured by a value like profit, cost, etc. Then, it is compared according to a certain criterion to have a total order of available options for decision-making. Finding the option with the highest or lowest value can be difficult work because the set of available options can be extremely large or not explicitly known (only the conditions that characterize the feasible options are known). A simple possible form to make a decision is having two alternative choices. An example is the *binary decision* that is formulated by binary variables "1" or "0". The value "1" means taking the first alternative, while the value "0" indicates that the second alternative is taken and the first one is rejected. The " 0/1 knapsack problem" is one of the most popular problems that consist of a binary decision process. Its name is derived from the problem faced to a mountaineer who is constrained by a limited capacity knapsack for a mountain tour and he has a large number of items that may be useful on his tour. The objective thus is to decide which items he should take with him by taking into account the limited capacity of the knapsack.

The 0/1 knapsack problem can be stated as follows: Given a set of N items, where each item i is characterized by a profit p_i, a weight w_i, and a knapsack having a limited capacity C. It is assumed that the profit and the weight of all items are non-negative integer numbers. The knapsack problem aims to select a subset of items to be packed into the knapsack, such that the total profit of the packed items is maximized and the total weight does not exceed the knapsack capacity. This latter can represent the maximum weight, the maximum available area or the limited height that can accommodate items, or even the maximum amount of items that can be carried. A generalized problem will get when more than one capacity constraint are taken in consideration, called "0/1 multidimensional knapsack problem" (0/1 MKP) or "d-dimensional knapsack problem", where d (>1) is the number of knapsack dimensions (or the number of capacity constraints).

This chapter is organized as follows. We begin by mentioning some useful concepts of optimization theory before giving a formal definition of 0/1 MKP in section 2. We then discuss several variations of 0/1 MKP in section 3. Thereafter, we describe some practical applications of 0/1 MKP and its variants in section 4, in order to highlight its usefulness in different fields. Finally, we conclude with a brief summary in section 5.

2 Problem definition

As stated earlier, the 0/1 multidimensional knapsack problem is a decision problem. More than that, it is a combinatorial optimization problem (Kellerer, et al., 2004) since it aims to find the best decision by making a combination (especially a linear combination) of the value

associated with each of the binary decisions. Before defining formally the problem, we recall some concepts of combinatorial optimization.

2.1 Basic concepts

An optimization problem aims to find an optimal solution of a given function, called an "objective function" or "cost function". Optimization problems can be restricted by some constraints that limit the search ability in the solution search space, giving rise to the so-called "constrained optimization problems". If no restrictions are required, the optimization problem is categorized as an "unconstrained optimization problem". The mathematical model of an optimization problem (Sierry & Collette, 2002) is given by:

$$\begin{cases} f(\vec{x}) \ (to\ be\ minimized\ or\ maximized\ ) & (1.1) \\ s.t \ \ \mathrm{g}(\vec{x}) \ \leq \ 0 \ (\text{or} \ \geq 0); \qquad (m\ constraints) & (1.2) \\ h(\vec{x}) \ = 0\ ; \qquad (p\ constraints) & (1.3) \\ \vec{x} \in S; & (1.4) \end{cases}$$

Where $\vec{x}$ is a vector solution, $\mathrm{g}(\vec{x}) \in \mathbb{R}^m$, and $h(\vec{x}) \in \mathbb{R}^p$. Equation (1.1) represents the objective function f. Equations (1.2) and (1.3) indicate respectively $m(\geq 1)$ inequality constraint(s) and $p(\geq 1)$ equality constraint(s). In general, the values of vector $\vec{x}$ verifying equation (1.4) define the "*solution search space*" or "*state space*" (*i.e* a set of all possible solutions), while the values of vector $\vec{x}$ verifying equations (1.2) and (1.3) define the "*search space of feasible solutions*".

Definition 1.1. Feasible solutions

A feasible solution is an alternative solution satisfying the entire constraints imposed by the optimization problem.

Definition 1.2. Objective function

The objective function f is an application $f: S \rightsquigarrow \mathbb{R}$ that associates, for each solution $\vec{x}$, an objective value $f(\vec{x})$. It aims to find an optimal solution (global optimum) among all feasible solutions. An objective function combines the decision variables by using a linear form (Sollow, 2007), a quadratic form (Christodoulos & Visweswaran, 1995), or a non-linear form (D'Ambrosio, 2010).

Definition 1.3. Decision variables

The decision variables are the elements x_i of the vector solution $\vec{x}$. The objective function can find its global optimum by varying the value of these decision variables. The nature of the values assigned to decision variables determines the kind of optimization problem:

- An optimization problem belongs to combinatorial optimization class (also called discrete optimization) whether $x_i \in \{1, \dots, N\} \subseteq \mathbb{Z}, \forall i$. In particular, it is considered as a binary optimization problem if $x_i \in \{0,1\}, \forall i$.
- An optimization problem belongs to continuous optimization class when the decision variables $x_i \in [a, b] \subseteq \mathbb{R}$, $\forall i$, where $a > b$.
- Many real-world optimization problems involve a mixture of continuous and discrete decision variables, giving rise to another kind of optimization called "mixed-variable optimization".

Definition 1.4. Global optimum

The vector solution $\vec{x}^*$ is evaluated as a global optimum if it is a feasible solution and verifies the following rule:

$$\forall \vec{x} \in \vec{X} \ \ and \ \vec{x} \neq \vec{x}^*: \begin{cases} f(\vec{x}^*) < f(\vec{x}); & (for\ minimization\ problem) \\ f(\vec{x}^*) > f(\vec{x}); & (for\ maximisation\ problem) \end{cases}; \qquad (1.5)$$

Where $\vec{X}$ is a set of all feasible solutions.

Definition 1.5. Local optimum

The vector solution $\vec{x}^*$ is evaluated as a local optimum if it is a feasible solution and verifies the following rule:

$$\forall \vec{x} \in V(\vec{x}^*) \ \ and \ \vec{x} \neq \vec{x}^* \ \begin{cases} f(\vec{x}^*) < f(\vec{x}); & (for\ minimization\ problem) \\ f(\vec{x}^*) > f(\vec{x}); & (for\ maximization\ problem) \end{cases}; \qquad (1.6)$$

Where $V(\vec{x}^*)$ is a neighborhood of the vector solution $\vec{x}^*$ that means the set of solutions close to $\vec{x}^*$, which is obtained by varying the value of some decision variables of $\vec{x}^*$. It is worth mentioning that the global optimum is considered as the best of local optima (see Figure 3).

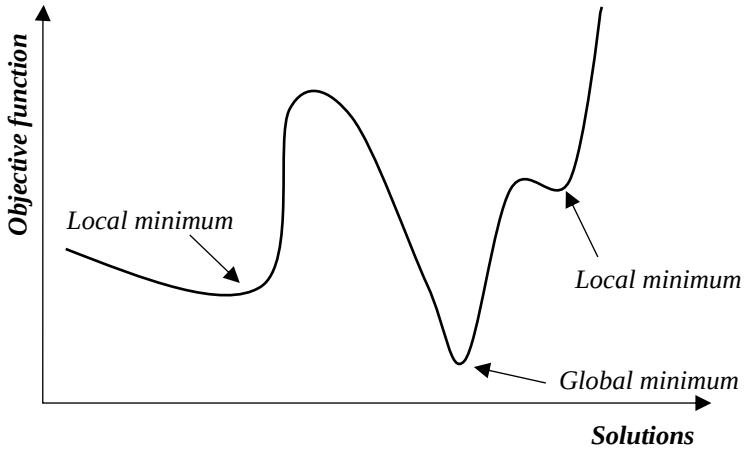


Figure 3. Example of Different minima

2.2 The 0/1 multidimensional knapsack problem

The 0/1 MKP is a classical combinatorial optimization problem, especially a binary optimization problem. It belongs to "Cutting & Packing problems", see the articles of ((Dyckhoff, 1990) and (Wäscher, et al., 2007)). The 0/1 MKP can be defined as following: Given a knapsack with m dimensions and a list of n items. Each item i has a profit $p_i > 0$ and a weight $w_{ij} \geq 0$ associated with the j-th dimension of the knapsack, where $i = 1, \dots, n$ and $j = 1, \dots, m$. Knowing that each dimension j has a limited capacity $C_j > 0$, the goal of the problem is to maximize the total profit of packed items into the knapsack while respecting its capacity constraints. It is assumed that, for all i and j, $w_{ij} < C_j$. The mathematical formulation of 0/1 MKP is expressed with an integer linear programming model[1]:

$$\begin{cases} Max \sum_{i=1}^{n} p_i x_i & (1.7) \\ s.t \sum_{i=1}^{n} w_{ij} x_i \leq C_j ; \quad \forall j = 1, \dots, m ; & (1.8) \\ x_i \in \{0,1\} ; \quad \forall i = 1, \dots, n; & (1.9) \end{cases}$$

Equation (1.7) provides the objective function (in a linear form) that consists of a linear combination of selected items' profits. Equation (1.8) indicates the capacity constraint of each knapsack dimension, whereas equation (1.9) means that x_i is a binary decision variable, it equals 1 if the i-th item is selected and 0 otherwise.

The 0/1 MKP has been intensively studied by attracting both researchers and practitioners. It is used by researchers as a test problem to evaluate the performance of their proposed algorithms. However, the practical interest arises mainly from its simple structure which, on one hand, can be used as a sub-problem to solve more complex problems, on the other hand, can model many real-world problems from industry, economics, and civil engineering, etc. Due to this variety

[1] In an integer linear programming, the decision variables are discrete, and the objective function as well as constraints are linear.

of applications, the 0/1 MKP can be found in the literature with different objective functions and additional constraints leading to many interesting variants.

3 Variants of 0/1 MKP

The 0/1 MKP can model a large number of practical applications by making some changes, without losing its specific structure, in terms of type and number of constraints, nature of decision variables, objective function, availability of input data, and so on. According to input data, we can classify the variants of 0/1 MKP into two different categories: *Deterministic variants* class in which the input data are known a priori, and *non-deterministic variants* class when the input data are imprecise, precisely, they are fuzzy or probabilistic. We note that fuzziness arises when we have a finite number of possibilities for input data, for going deeper see the reference book of (Klir & Yuan, 1995).

3.1 Deterministic variants of 0/1 MKP

3.1.1 Multiple MKP

The 0/1 Multiple MKP (0/1 MMKP) differs from the 0/1 MKP in terms of the number of knapsacks. Instead of a single knapsack, $M(>1)$ knapsacks are considered, each has d dimensions with limited capacity ${C_j}^k$, where $k = 1,..,M$ and $j = 1,\dots,m$. Moreover, each item i ($\in \{1,\dots,n\}$) has a profit p_i and a weight ${w_{ij}}^k$ associated with the j-th dimension of the knapsack k. It should be emphasized that the decision here is not only which item to select but also to define in which knapsack it will be packed. Mathematically, the 0/1 MMKP is given by:

$$\begin{cases} Max \quad \sum_{k=1}^{M}\sum_{i=1}^{n} p_i x_{ik} & (1.10) \\ s.t \sum_{i=1}^{n} {w_{ij}}^k \, x_{ik} \; \leq {C_j}^k; \quad \forall\, j = 1,\dots,m\,;\ \forall\, k = 1,\dots,M; & (1.11) \\ \sum_{k=1}^{M} x_{ik} \leq 1\,; \quad \forall i = 1,\dots,n; & (1.12) \\ x_{ik} \in \{0,1\}\,; \quad \forall\, i = 1,\dots,n\,;\ k = 1,\dots,M; & (1.13) \end{cases}$$

Equation (1.10) provides the total profit of packed items into available knapsacks. Equation (1.11) ensures that the capacity constraints of each knapsack are respected, while equation (1.12) ensures that each item appears at most once in all knapsacks. Finally, equation (1.13) represents the binary decision variables x_{ik} such that ($x_{ik} = 1$) if the i-th item is selected and packed into k-th knapsack, and ($x_{ik} = 0$) otherwise.

3.1.2 Multiple Choice MKP

The 0/1 Multiple-Choice MKP (0/1 MCMKP) is a variant of 0/1 MKP in which n items are divided into N disjoint groups[2] and each group k contains N_k items. Besides, each item i of the group k has a profit p_{ik} and a weight ${w_{ik}}^j$ associated with the j-th dimension of the knapsack, where $i \in \{1, \dots, N_k\}$, $k \in \{1, \dots, N\}$, and $j \in \{1, \dots, m\}$. As it is common, each dimension j of the knapsack has a capacity C_j. The 0/1 MCMKP aims to select only one item from each group such that the total profit of the knapsack is maximized and its capacity constraints are respected. Formally, the 0/1 MCMKP can be stated as follows:

$$\begin{cases} Max \sum_{k=1}^{N} \sum_{i=1}^{N_k} p_{ik} x_{ik} & (1.14) \\ s.t \sum_{k=1}^{N} \sum_{i=1}^{N_k} {w_{ik}}^j \; x_{ik} \leq C_j \; ; \quad \forall j = 1, \dots, m; & (1.15) \\ \sum_{i=1}^{N_k} x_{ik} = 1; \quad \forall k = 1, \dots, N; & (1.16) \\ x_{ik} \in \{0,1\}; \quad \forall k = 1, \dots, N \; ; \; i = 1, \dots, N_k; & (1.17) \end{cases}$$

Equation (1.14) provides the total profit of items selected from each group. Equation (1.15) ensures that the capacity constraints are simultaneously satisfied. Besides, equation (1.16) indicates that only one item is selected from each group. Finally, equation (1.17) represents the binary decision variables x_{ik} such that ($x_{ik} = 1$) if the item i of the group k is selected and ($x_{ik} = 0$) otherwise.

3.1.3 Multi-objective MKP

Some optimization problems involve more than one objective function, giving rise to another class of optimization, called "multi-objective optimization" (see the reference book of (Sierry & Collette, 2002)). An example of this kind of problems is the multi-objective MKP (0/1 MOMKP), which is a variant of 0/1 MKP wherein K (> 1) conflicting objective functions have to be optimized. Given a knapsack having m dimensions and a set of n items. Each item i has a profit ${p_i}^k$ related to the k-th objective function, and a weight w_{ij} associated with the j-th dimension of the knapsack. As the 0/1 MKP, each dimension j of the knapsack has a capacity C_j. The goal is to select a subset of items that meet the objective function subject to knapsack capacity constraints. The mathematical formulation of 0/1 MOMKP (Schulze, 2017) is defined below:

[2] Disjoint groups verify : $\forall k, k' \in \{1,2,..,N\}$ so that $k \neq k'$, $N_k \cap N_{k'} = \emptyset$ and $n = \cup_{k=1}^{n} N_k$

$$\begin{cases} Max\, z_k = \sum_{i=1}^{n} p_i^k\, x_i \quad \forall\, k = 1, \dots, K; & (1.18) \\ s.t \sum_{i=1}^{n} w_{ij}\, x_i \leq C_j\,; \quad \forall\, j = 1, \dots, m; & (1.19) \\ x_i \in \{0,1\}\,; \quad \forall\, i = 1, \dots, n; & (1.20) \end{cases}$$

Equation (1.18) calculates the total profit of selected items for each objective function, while equation (1.19) ensures that capacity constraints are satisfied. Finally, equation (1.20) means that x_i is a binary decision variable, it equals 1 if the item i is selected and 0 otherwise.

3.2 Non-deterministic variants of 0/1 MKP

3.2.1 Stochastic MKP

In a stochastic version of 0/1 MKP, the information about the weight of items is uncertain. However, it is assumed that the weights are independent random variables following given probability distribution. As the deterministic 0/1 MKP, the objective function consists of maximizing the total profit of the knapsack. It is worth mentioning that unlike the weight of items, the information about their profit is available. For the sake of clarity, the weight of item i is denoted by ξ_{ij} instead of w_{ij}. The stochastic 0/1 MKP is formulated by (Kunikazu & Prékopa, 2012) as probabilistic-constrained stochastic programming[3]:

$$\begin{cases} Max \sum_{i=1}^{n} p_i\, x_i & (1.21) \\ Pr\left(\sum_{i=1}^{n} x_i\, \xi_{ij} \leq C_j \right) \geq q\,; \quad \forall\, j = 1, \dots, m; & (1.22) \\ x_i \in \{0,1\}\,; \quad \forall\, i = 1, \dots, n; & (1.23) \end{cases}$$

Where $q \in [0,1]$ is a preset probability value. The equation (1.21) provides the total profit of selected items. Equation (1.22) points out the probabilistic constraints imposed by the stochastic context of the problem. Finally, equation (1.23) represents the binary decision variable x_i that indicates whether the item i is selected or not.

By assuming that the random variables ξ_{ij} are independent, the probabilistic constraints represented by equation (1.22) can be rewritten as:

$$\prod_{j=1}^{m} Pr\left(\sum_{i=1}^{n} x_i\, \xi_{ij} \leq C_j \right) \geq q \qquad (1.24)$$

[3] Also called chance-constrained stochastic programming, it is a mathematical formulation of an optimization problem ensuring that the probability of meeting constraints is above a predetermined probability value.

3.2.2 Fuzzy MKP

When the information about weight or/ and profit of items are ambiguous, the fuzzy set theory is applied to deal with this ambiguity, which leads to a fuzzy variant of 0/1 MKP (0/1 FMKP). This latter can be stated as follows:

$$\begin{cases} Max \sum_{i=1}^{n} \widetilde{p}_i \, x_i & (1.25) \\ \sum_{i=1}^{n} \widetilde{w}_{ij} \, x_i \leq C_j ; \quad \forall j = 1, \dots, m; & (1.26) \\ x_i \in \{0,1\} ; \quad \forall i = 1, \dots, n; & (1.27) \end{cases}$$

Let $\widetilde{p}_i$ and $\widetilde{w}_{ij}$ denote fuzzy intervals related, respectively, to the profit of item *i* and the weight of item *i* in the *j*-th dimension. Equation (1.25) provides the total profit of selected items, while equation (1.26) ensures that the capacity constraints are satisfied. Finally, the binary decision variable x_i in equation (1.27) indicates whether the item *i* is selected or not.

4 Real-world applications

The 0/1 MKP has firstly appeared in the economical context of capital rationing (Lorie & Savage, 1955). In the last decades, the 0/1 MKP and its variations have been intensively used by the decision-makers in order to model faced optimization problems in real-life situations. In the following subsections, we give some examples issued from different fields like economics, telecommunications systems, distributed computing system, and civil engineering.

4.1 Multi-unit combinatorial auctions

The Combinatorial Auctions (CAs) consist of a publicly held sale wherein the seller places competitive properties, called "bids" in the bidding language, which should be sold to the highest bidder[4]. A prominent variant of CAs is Multi-Unit Combinatorial Auctions (MUCA). It differs from CAs or more precisely Single-Unit CAs in the number of copies of each property: The Single-Unit CAs have only one unit per propriety. MUCA can be formally defined as following. The auctioneer has a set of *d* properties to sell. Each property *j* is available in a certain quantity of copies, where $j = 1, \dots, m$. Given a list of *n* submitted bidders, each offers a price $p_i > 0$ and requests a number of copies r_{ij} of the property *j*, where $i = 1, \dots, n$. The objective is selecting bidders that offer the highest prices in order to maximize the auctioneer's revenue. MUCA is mathematically formulated by ((Pfeiffer & Rothlauf, 2007) and (Sandholm, et al., 2002)) by means of 0/1 MKP model, in which the bidders are regarded as items in the

[4] The person who offers the most money for a property

context of 0/1 MKP, while the properties represent the knapsack dimensions. The mathematical formulation defined as bellow:

$$\begin{cases} Max \sum_i p_i \, x_i & (1.28) \\ s.t \sum_i r_{ij} \, x_i \leq u_j \, ; \quad \forall \, j = 1, \dots, m & (1.29) \\ x_i \in \{0,1\} \, ; \quad \forall \, i = 1, \dots, n & (1.30) \end{cases}$$

Equation (1.28) provides the total revenue of the auctioneer. Equation (1.29) ensures that the available quantity of copies of each property j is met. Finally, equation (1.30) represents the decision variable x_i that equals 1 if the i-th bidder is selected by the auctioneer and it equals 0 otherwise.

4.2 Resources allocation with stochastic demands

A distributed computing system refers to multiple computers working on the same task, which is divided into many parts that are simultaneously accomplished by different users. The goal of a distributed computing system is to maximize its efficiency and performance by giving its users the necessary resources. However, the allocation of resources to users is a vital problem in such system. This problem can be defined as following: Given a list of m resources and a set of n users. Each resource j ($\in \{1, \dots, m\}$) is available in a limited quantity, and each user i ($\in \{1, \dots, n\}$) is associated with a profit c_i and requests an amount A_{ij} of the resource j. The objective is satisfying the users' demands as maximum as possible taking into account the resource limitations. (Chen, et al., 2012) have studied the problem of resource allocation with stochastic demands (*i.e.* the users' demands follow a given probability distribution). They have modeled this problem by stochastic 0/1 MKP wherein the resources correspond to knapsack dimensions, while the users correspond to items to be packed into knapsack. The model is given by:

$$\begin{cases} Max \sum_i c_i \, x_i & (1.31) \\ s.t \; Pr\left(\sum_i A_{ij} \, x_i > b_j\right) \leq p \, ; \quad \forall \, j = \{1, \dots, m\} \, ; & (1.32) \\ x_i \in \{0,1\} \, ; \quad \forall \, i = \{1, \dots, n\} \, ; & (1.33) \end{cases}$$

Equation (1.31) provides the total profit of admitted users (*i.e.* the users who are selected to satisfy their demands). Equation (1.32) indicates the probabilistic constraints. Finally, equation (1.33) represents the binary decision variable x_i that indicates whether the demand of the j-th user is satisfied or not.

4.3 Frequency allocation in cognitive radio networks

The cognitive radio network (CRN) consists of assigning electromagnetic spectrum to frequency bands in a dynamic manner. The CRN distinguishes between two types of cognitive users: Primary users that are licensed to operate in certain frequency bands (called primary bands) and secondary users that make dynamic use of the spectrum in such a way they do not cause interference to primary users. The main goal of such frequency allocation is to exploit the spectrum as maximum as possible. (Song, et al., 2008) have formulated the frequency allocation in cognitive radio networks by using the 0/1 MMKP model. They consider the CRN that has a Centralized Coordinator Node[5] (CCN) and a certain number of cognitive users, where each cognitive user is a pair of transmitter and receiver. They admit that there is a common channel for communication between cognitive users and CCN. Given a list of n cognitive users and a list of M primary users. Each cognitive user i requests a bandwidth d_i from CCN and generates an interference level I_{ik} to the primary user k. Let B_k and K_k denote, respectively, the bandwidth and the interference temperature[6] of the primary user k. The mathematical formulation proposed by authors is given by:

$$\begin{cases} Max \sum_{k=1}^{M} \sum_{i=1}^{n} h(d_i)\, x_{ik} & (1.34) \\ s.t \sum_{i=1}^{n} d_i\, x_{ik} \le B_k\,; \quad \forall\, k = 1, \dots, M; & (1.35) \\ \sum_{i=1}^{n} I_{ik}\, x_{ik} \le K_k\,; \quad \forall k = 1, \dots, M; & (1.36) \\ \sum_{k=1}^{M} x_{ik} \le 1\,; \quad \forall i = 1, \dots, n; & (1.37) \\ x_{ik} \in \{0,1\}\,; \quad \forall i = 1, \dots, n\,;\ k = 1, \dots, M; & (1.38) \end{cases}$$

Equation (1.34) provides the overall throughput of the spectrum. Notice that the throughput is considered as a function of bandwidth, denoted h. Equation (1.35) ensures that the bandwidth of each primary user is not exceeded, whereas equation (1.36) ensures that the accumulated interference at each primary user band is kept below its interference temperature. Equation (1.37) indicates that the bandwidth request of each cognitive user is satisfied by assigning it at most one primary user band. Finally, equation (1.38) means that x_{ik} is a decision variable, which equals 1 if the bandwidth request of the i-th cognitive user is satisfied by allocating to it the k-th primary user band, and it equals to 0 otherwise. To keep things clear and simple, the cognitive users are considered as items to be packed into the knapsack, while the primary users are regarded as the available knapsacks. Besides, as described hereinabove, there are two

[5] The CCN collects the transmission requests from all cognitive users and allocates the available spectrum optimally so that the overall throughput is maximized.

[6] The interference threshold accepted by the primary user

dimensions of the knapsack that correspond to the bandwidth and the interference temperature of primary users.

4.4 MPSoC runtime management

MultiProcessor System On Chip (MPSoC) is a multiprocessor platform in which several applications can be simultaneously activated. The major challenge of MPSoC is minimizing the energy consumption of applications at runtime, by respecting the resource limitations and the applications deadlines. Knowing that each application has different possible implementations, (Ykman-Couvreur, et al., 2001) have modeled the current optimization problem, which they called "MPSoC runtime management", by means of the 0/1 MCMKP model. Given a set of n active applications, each contains N_k operating points[7], where $k \in \{1, \dots N\}$, and a set of m resources (*e.g.* memory space, number of processors…). Each operating point i of the active application k runs in t_{ik} time by consuming an energy e_{ik}, and requires a quantity ${r_{ik}}^j$ of the resource j ($\in \{1, \dots, m\}$). For the sake of clarity, each active application is regarded as a group of items. The MPSoC corresponds to the knapsack, while the type of resources and the deadline of applications are considered as the knapsack dimensions. The mathematical formulation of the problem is given by:

$$\begin{cases} Max \sum_{k=1}^{N} \sum_{i=1}^{N_k} v_{ik} x_{ik} & (1.39) \\ s.t \sum_{k=1}^{N} \sum_{i=1}^{N_k} x_{ik} \, {r_{ik}}^j \leq R_j ; \quad \forall j = 1, \dots, m; & (1.40) \\ \sum_{i=1}^{N_k} x_{ik} \, t_{ik} \leq D_k ; \quad \forall k = 1, \dots, N; & (1.41) \\ \sum_{i=1}^{N_k} x_{ik} = 1 ; \quad \forall k = 1, \dots, N ; & (1.42) \\ x_{ik} \in \{0,1\} ; \quad \forall i = 1, \dots, N_k ; \; k = 1, \dots, N ; & (1.43) \end{cases}$$

Note that $v_{ik} = e_{1k} - e_{ik}$, $\forall k \in \{1, \dots, N\}$ and $i = 1, \dots, N_k$, where e_{1k} means the maximum energy consumption of the active application k. Besides, v_{ik} is a non-negative value since each active application is considered as an ordered set according to the decreasing order of the energy consumed by operating points.

Equation (1.39) provides the total energy consumption of selected operating points. Equation (1.40) ensures that the required resources don't exceed the available resources, while equation (1.41) indicates that the execution time of each selected point from set k must not exceed the application deadline D_k. Equation (1.42) indicates that at most one operating point is selected

[7] They are defined in a search space having multiple dimensions (resources, consumed energy,…). They characterize optimal application mappings on the platform (for more details see (Ykman-Couvreur, et al., 2001)).

from each active application. Finally, the binary decision variable x_{ik} in equation (1.43) denotes whether the point *i* is selected from the set *k* or not.

4.5 Capital budgeting problem

The capital budgeting problem consists of selecting investment projects that maximize the profit of a company, taking into account a limited budget. (Khalili-Damghani & Taghavifard, 2011) have been interested in the capital budgeting problem in a fuzzy environment. They have modeled it as a 0/1 FMKP aiming at two key objectives: Lowest investment cost and maximum profit. Given a set of n projects, each project j procures a profit p_j, for $j = 1, \dots, n$, and requires $\tilde{h}_{ij}$ human resources of type i ($\in \{1,2, \dots, r\}$), $\tilde{m}_{kj}$ machines per hour of type k ($\in \{1,2, \dots, s\}$) as well as $\tilde{r}_{oj}$ raw materials of type o ($\in \{1,2, \dots, z\}$). Let $\tilde{C}_i$, $\tilde{C}_k$, and $\tilde{C}_o$ denote, respectively, the cost per hour of human resource i, cost per hour of machine type k, and per-unit cost of raw material o. Note that notations under tilde represent fuzzy parameters. For the record, the projects are considered as items to be packed into the knapsack, while the resources type and the capital budget correspond to dimensions of the knapsack. The formulation of the problem is showed below:

$$
\begin{cases}
Max \sum_{j=1}^{n} x_j \left[p_j - \sum_{i=1}^{r} \tilde{h}_{ij}\, \tilde{C}_i - \sum_{k=1}^{s} \tilde{m}_{kj}\, \tilde{C}_k - \sum_{o=1}^{z} \tilde{r}_{oj}\, \tilde{C}_o \right] & (1.44) \\
s.t \sum_{j=1}^{n} \tilde{h}_{ij}\, x_j \le \tilde{H}_i ; \quad \forall\, i = 1, \dots, r; & (1.45) \\
\sum_{j=1}^{n} \tilde{m}_{kj}\, x_j \le \tilde{M}_k ; \quad \forall\, k = 1, \dots, s; & (1.46) \\
\sum_{j=1}^{n} \tilde{r}_{oj}\, x_j \le \tilde{R}_o ; \quad \forall\, o = 1, \dots, z; & (1.47) \\
\left(\sum_{i=1}^{r} \tilde{h}_{ij}\, \tilde{C}_i + \sum_{k=1}^{s} \tilde{m}_{kj}\, \tilde{C}_k + \sum_{o=1}^{z} \tilde{r}_{oj}\, \tilde{C}_o \right) x_j \le \tilde{B}_j; \quad \forall j = 1, \dots, n; & (1.48) \\
\left(\sum_{i=1}^{r} \tilde{h}_{ij}\, \tilde{C}_i + \sum_{k=1}^{s} \tilde{m}_{kj}\, \tilde{C}_k + \sum_{o=1}^{z} \tilde{r}_{oj}\, \tilde{C}_o \right) x_j < \tilde{p}_j; \quad \forall\, j = 1, \dots, n; & (1.49) \\
\sum_{j=1}^{n} x_j \ge 1 ; & (1.50) \\
x_j \in \{0,1\}; \quad \forall\, j = 1, \dots, n; & (1.51)
\end{cases}
$$

Equation (1.44) provides the total net profit of selected projects. Equations (1.45), (1.46), and (1.47) indicate that the resource constraints concerning, respectively, human resources, machines, and raw materials are met. Equations (1.48) ensures that the budget $\tilde{B}_j$ devoted to each selected project j is respected, while equation (1.49) means that the investment cost of each selected project is less than its profit. Finally, equation (1.50) indicates that at least one

project should be selected and equation (1.51) represents the decision variable x_j that denotes whether the project *j* is selected or not.

4.6 Real estate property maintenance problem

The real estate property maintenance problem (REPMP) consists of maintaining various real estate components that are part of buildings in smooth running condition. There are two types of maintenance: Common maintenance that is devoted to small operations (*e.g.* elevator out of service) usually performed according to predefined procedures. The second type is planned maintenance devoted to heavier operations (*e.g.* facade renovating, change of heating system), which is generally subject to a specific multi-year action plan[8]. The main goal of REPMP aims to find an optimal multi-year plan to carry out in a limited period while respecting a limited budget. (Taillandier, et al., 2017) have tackled this problem as being 0/1 MOMKP. Given a set of n maintenance actions and a certain number m of years that the maintenance of the real estate property lasts. Each action *i* has an execution cost w_{ij} during the *j*-th year of the maintenance plan and a profit c_{ij}^k at the *j*-th year regarding the objective k, where $i = 1, \dots n$, $k = 1, \dots, K$, and $j = 1, \dots m$. Indeed, a score relative to an objective k is assigned to every component of buildings. A lower score corresponds to the worst situation in which a component has a major failure in a given objective k, whilst a higher score corresponds to its best situation. Thus, the profit of each action corresponds to its capacity to reduce the difference between the required score and the actual score of components. For ease of reference, the actions are considered as items to be packed into the knapsack, while the several years of the maintenance plan correspond to knapsack dimensions. The mathematical formulation of this maintenance problem is given by:

$$\begin{cases} Max\, z_k = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}^k\, x_{ij}\,; \quad \forall k = 1, \dots, K; & (1.52) \\ s.t \sum_{i=1}^{n} w_{ij} x_{ij} \le W_j\,; \quad \forall j = 1, \dots, m; & (1.53) \\ \sum_{j=1}^{m} x_{ij} \le 1; \quad \forall i = 1, \dots, n; & (1.54) \\ x_{ij} \in \{0,1\}; \quad \forall\ i = 1, \dots, n\,;\ j = 1, \dots, m; & (1.55) \end{cases}$$

Equation (1.52) provides the global profit of selected actions for each objective, whereas equation (1.53) ensures that the total cost of the actions executed in every year must not exceed the allocated yearly budget. Equation (1.54) ensures that each action is executed only once during the multi-year maintenance plan. Finally, equation (1.55) represents the binary variable

[8] A multi-year action plan is a plan outlining maintenance actions needed to achieve a long-term vision

decision x_{ij} that indicates whether the action i is executed in the j-th year of the maintenance plan or not.

5 Conclusion

In this paper, the literature about the binary version of the multidimensional knapsack problem (0/1 MKP) is reviewed. It is formally described as well as many relevant variants are introduced, in addition to an interesting collection of recently published real-world applications of 0/1 MKP and its variants.

Chapter 2: Resolution methods of 0/1 MKP

1 Introduction

According to the theory of complexity that aims to study the complexity of problems in terms of memory space and execution time, the decision and optimization problems are classified into two classes: P class (P for Polynomial time) and NP class (NP for Non-deterministic Polynomial time). The first class contains easy problems that can be optimally resolved in polynomial time regardless of the problem instance[9] size. The second class contains significantly hard problems for which we cannot found any algorithm that optimally solves them in polynomial time, in particular, when the problem size becomes quite large. Moreover, the NP class distinguishes between two subclasses: NP-complete problems that contain only decision problems, and NP-hard problems including both decision and optimization problems. It should be emphasized that whether the decision problem associated to an optimization problem is an NP-complete problem, this latter is automatically considered as an NP-hard problem. However, the relationship between NP and P belongs to the seven millennium problems that are still unsolved until this day. Such classification is very important and should be taking into account in order to select the appropriate algorithm for solving the problem at hand.

Despite the straightforward statement as well as the easy mathematical formulation of 0/1 MKP, it belongs to NP-hard combinatorial optimization problems since it is a generalization of the 0/1 knapsack problem that is already proved to be NP-hard (Karp, 1972). As a result, a wide variety of approximate algorithms are developed to deal with 0/1 MKP that require reasonable time to obtain a good quality of solutions but without ensuring the optimality. This chapter provides a survey of existing resolution methods of 0/1 MKP. Section 2 provides an overview of exact methods developed to optimally solve 0/1 MKP, while sections 3 and 4 are devoted to heuristic approaches that seek to find a tradeoff between solution quality and execution time. Finally, a brief conclusion is given in section 5.

2 Exact methods

The 0/1 MKP is significantly harder to be resolved by employing exact methods. As the number of knapsack constraints and the number of items increase, exact methods fail to provide an optimal solution in a reasonable runtime. That is why only a few exact methods are addressed to solve 0/1 MKP. The development of these methods began during the sixties. One of the early exact methods is the dynamic programming algorithm proposed by (Gilmore & Gomory, 1966). Then, (Green, 1967) has proposed extensions of the Gilmore and Gomory's algorithm. Thereafter, (Bertsimas & Demir, 2002) have developed an approximate dynamic programming (ADP) approach that uses a heuristic based upon the linear relaxation of 0/1 MKP. The experiments illustrated that their ADP approach competes successfully with state-of-the-art

[9] "instances" means the input data of the problem at hand.

heuristics and the commercial software CPLEX. Another exact method has been proposed to solve 0/1 MKP, it is the recursive branch and bound algorithm that was published by (Thesen, 1975). The computational results demonstrated that the proposed algorithm is competitive for small to medium instances, but it is impractical for instances containing up to 100 variables. One more branch and bound algorithm was published by (Shih, 1979). Their proposed algorithm was tested on a set of randomly generated instances containing up to 90 items and 5 knapsack dimensions, and compared with the general zero-one additive algorithm of (Balas, 1965). The obtained results have proved its superiority. Furthermore, an efficient branch and bound algorithm was developed by (Gavish & Pirkul, 1985) using various relaxations of the problem, including Lagrangian, surrogate, and composite relaxations. Their algorithm was proved to be faster than the one proposed by (Shih, 1979) for randomly generated instances with up to 200 items and 5 knapsack dimensions. It was also compared to, and found to be faster than, the solver SCICONIC/VM[10]. Other kinds of exact algorithms with only limited success being reported: (Cabot, 1970) has suggested an enumeration method by using the Fourier-Motzkin elimination. The experiments were carried out on a set of 50 randomly generated instances with 10 variables and 5 knapsack dimensions. The results are somewhat scanty. (Soyster, et al., 1978) have proposed an iterative scheme in which linear programs were solved in order to generate sub-problems, which were then solved via an implicit enumeration strategy. The experiments have shown that the proposed algorithm is convenient to resolve 0/1 MKP for test problems including 50 to 400 variables and 5 to 10 knapsack dimensions. The idea of tackling 0/1 MKP by solving a sequence of sub-problems was also used by (Vasquez & Hao., 2001) and (James & Nakagawa, 2005) in an exact approach.

Lately, (Vimont, et al., 2008) have proposed an implicit enumeration for solving 0/1 MKP based on the idea that the unpromising parts of the search tree should be tackled first. Inspired by this exact algorithm, (Mansini & Speranza, 2012) have introduced an exact approach called CORAL. The experiments were carried out on two different classes of 0/1 MKP benchmark instances. The first class consists of 270 large instances, while the second considers 7 instances out of 11 more large instances. The results have proved the efficiency of CORAL in comparison with the algorithm of (Vimont, et al., 2008) as well as CPLEX 10[11]. Similarly, (Boussier, et al., 2009) have proposed a new exact method that combines the resolution search algorithm of (Chvátal, 1997) with a branch and bound algorithm, by inspiring from the idea of (Vimont, et al., 2008). They solved to optimality all instances with 500 variables and 10 knapsack dimensions by requiring a long average time. The combination of the resolution search algorithm with a branch and bound algorithm has also been used by (Boussier, et al., 2010) in a more efficient way. The experiments have proved that the hybrid method is able to solve optimally large-scale instances containing 500 items and 10 constraints, and others including

[10] A commercial mixed-integer programming solver that was developed by (Scicon, 1986)

[11] An optimization software developed by Robert E. Bixby in 1987 and continues to be actively developed by IBM.

250 items and 30 constraints. However, their approach requires a long runtime for up to several days.

3 Heuristics

Heuristic algorithms for solving 0/1 MKP are reviewed below by clustering heuristics having the same principles.

3.1 Greedy heuristics

The greedy heuristics (or constructive heuristics) are easy to implement and need a few times to build approximate solutions. The 0/1 MKP literature distinguishes between two basic greedy heuristics: Primal greedy heuristic ((Kochenberger, et al., 1974), (Toyoda, 1975), and (Loulou & Michaelides, 1979)) that begin with a pre-processing procedure wherein all items are sorted according to a certain criterion (profit, weight, or the combination of both) in decreasing order. Then, it packs items in turn into the knapsack as long as capacity constraints are met. The main steps of a primal greedy heuristic are presented in algorithm 1. On the other side, the dual greedy heuristic (Senju & Toyoda, 1968) that starts the other way around. It first sets all decision variables to one, then sorts items in increasing order according to a certain criterion, and finally removes items iteratively as long as capacity constraints are violated. In a final step, all removed items must be considered again for inclusion in order to improve the quality of the obtained solution. This post-processing step consists of sorting removed items so that the most valuable items should be included first.

1. **Input: List of n items and a knapsack with m dimensions**
2. Initialize $R_j = \sum_{i=1}^{n} w_{ij}\, x_i$ (R_j is the accumulated resources of the constraint j)
3. For $i = 1{:}\,n$
4. Sort f_i's in decreasing order as well as their corresponding items
5. (f_i is a given criterion for sorting items)
6. While $R_j \leq C_j\,,\ \forall j = 1,\ \ldots,\ m$
7. If $R_j + w_{ij} \leq C_j$ and $x_i = 0$ then
8. $x_i = 1$
9. $R_j = R_j + w_{ij}\,,\ \ \forall j = 1,\ \ldots,\ m$
10. $i = i + 1$
11. End For
12. **Output: Feasible solution of 0/1 MKP**

Algorithm 1. The pseudo-code of a primal greedy heuristic

In recent past, (Volgenant & Zwiers, 2007) have developed a new heuristic to solve 0/1 MKP, which partially enumerates the search space instead of enumerating all variables. The simulations showed that the proposed heuristic improves the quality of solutions by consuming a modest computing time. (Akin, 2009) has introduced a new greedy heuristic, called sliding enumeration, that evaluates a group of variables instead of evaluating only one variable at a time. His heuristic performs better when the variables are ordered by an efficient strategy. (Akcay, et al., 2007) have adapted a greedy heuristic to 0/1 MKP, which is originally introduced for the integer MKP. The proposed heuristic yields good results in a short time for instances

containing 50 to 100 variables and 10 to 200 constraints. However, when the number of variables or constraints increases, it loses its dominance. (Drake, et al., 2014) have used genetic programming (GP) as a hyper-heuristic methodology to generate greedy heuristics to solve 0/1 MKP. The proposed hyper-heuristic operates on a search space of heuristics instead of a search space of solutions. It selects and applies a heuristic at each stage of the search process. The obtained results over a set of standard benchmarks showed that GP can be used to successfully generate the convenient greedy heuristic for solving 0/1 MKP.

3.2 Relaxation-based heuristics

The main objective of relaxation techniques is allowing obtaining, at the lowest computational time, tight upper bounds of the problem at hand. For 0/1 MKP, relaxations are incorporated in heuristics in order to generate feasible solutions that consist of optimal solutions obtained by using either LP relaxation or achieved indirectly by adopting Lagrangian or surrogate relaxations. This type of relaxations is also required in reduction procedure[12] for the elimination of redundant constraints, fixing variables, or generation of logical relations.... Besides, they are needed in partial enumeration for pruning nodes of the enumeration tree. The relaxation based-heuristics devoted to solving 0/1 MKP are nevertheless presented separately in the following sub-sections.

3.2.1 Linear programming relaxation

Linear programming (LP) relaxation, called also continuous relaxation, can be generated from 0/1 MKP by taking the same objective function and same constraints but with the requirement that variables are integer. Thus getting the following problem:

$$\begin{cases} Max \ \sum_{i=1}^{n} p_i \, x_i & (2.1) \\ s.t \ \sum_{i=1}^{n} w_{ij} \, x_i \ \leq \ C_j \, ; \quad \forall \, j = 1, \dots, m \, ; & (2.2) \\ 0 \leq \ x_i \leq 1 \ ; \quad \forall \, i = 1, \dots, n; & (2.3) \end{cases}$$

(Balas & Martin, 1980) have developed an LP relaxation-based heuristic that consists firstly of solving the linear program, then it performs a sequence of pivots aimed at putting all slacks into the basis at a minimal cost. Finally, a local search procedure based on complementing certain sets of binary variables is applied to improve the quality of the obtained solution. The simulations were reported for some randomly generated instances as well as some real-world instances, containing up to 200 constraints and up to 900 variables. Thus, (Lee & Guignard, 1988) have presented an approach that combines the heuristic of (Toyoda, 1975) with variables fixing, linear programming, and the complementing procedure of (Balas & Martin, 1980). The

[12] An algorithm that transforms problem A into a problem B so that the transformation takes polynomial time, and the answer for A is yes whether the answer for B is yes. Furthermore, if B can be solved in polynomial time, then A should be solved in polynomial time and if A is "hard", then B should be hard too (for more details, see (Cormen, et al., 2009)). In the context of 0/1 MKP, the reduction procedure involves removing some variables from the problem formulation or at least fixing those variables to some predetermined values.

simulations showed that the proposed heuristic performs well in comparison with those of (Toyoda, 1975) and (Magazine & Oguz, 1984), but it is surpassed by that of (Balas & Martin, 1980). Additionally, (Fleszar & Hindi, 2009) have suggested several fast and effective heuristics based on solving the linear programming relaxation of 0/1 MKP. The experiments demonstrated the effectiveness of the proposed heuristics against best-performing ones available in the literature by using the benchmark instances of 0/1 MKP. On the other side, (Hanafi & Wilbaut, 2011) have proposed an efficient heuristic approach for solving 0/1 MKP, by inspiring from the idea of (Soyster, et al., 1978) that consists of solving a series of small sub-problems generated by exploiting information obtained through a series of LP relaxations.

3.2.2 Lagrangian relaxation heuristics

The Lagrangian relaxation consists of dividing constraints into two categories: Easy and difficult constraints. The idea is to integrate the difficult constraints in the objective function with a certain penalty to generate an easier problem, see below:

Let $Q \subseteq \{1, \dots, m\}$, $q = |Q| \neq 0$, $\lambda = (\lambda_j)_{j \in Q} \in \mathbb{R}_+^q$

$$\begin{cases} Max \ \sum_{i=1}^{n} (p_i - \sum_{j \in Q} \lambda_j w_{ij}) x_i + \sum_{j \in Q} \lambda_j C_j & (2.4) \\ s.t \sum_{i=1}^{n} w_{ij} x_i \leq C_j ; \quad \forall j \in \{1, \dots, m\} \backslash Q ; & (2.5) \\ x_i \in \{0,1\} ; \quad \forall i = 1, \dots, n; & (2.6) \end{cases}$$

(Magazine & Oguz, 1984) have developed a heuristic combining the idea of (Senju & Toyoda, 1968) with the generalized Lagrange multiplier approach proposed by (Everett, 1963). Their proposed heuristic was tested upon a large randomly generated instances ranging from 20 to 1000 constraints and from 20 to 1000 items, then it was compared against the heuristics of (Kochenberger, et al., 1974) and (Senju & Toyoda, 1968). However, (Volgenant & Zoon, 1990) have proposed two improved versions of the heuristic proposed by (Magazine & Oguz, 1984). In recent past, (Yoon & Kim, 2013) have introduced a heuristic in which the Lagrangian multipliers are calculated by using a memetic algorithm[13]. The experimental results showed that the proposed method outperforms other constructive heuristics and local improvement heuristics for a variety of test instances. Besides, (Atılgan & Nuriyev, 2012) have proposed a heuristic in which the Lagrange multipliers are determined for every constraint to reduce the original problem to a single constraint problem, then the obtained solution is improved through iterative procedures. The simulations demonstrated the efficiency of the proposed heuristic for some standard instances. On the other side, (Yoon, et al., 2012) have developed an efficient

[13]A genetic algorithm using a local search procedure instead of using the mutation strategy.

heuristic based on the concept of Lagrangian capacity (*i.e.* the value of the capacity constraint allowing Lagrangian relaxation to find an optimal solution).

3.2.3 Surrogate relaxation heuristics

An alternative way to relax 0/1 MKP is through a surrogate approach. The surrogate constraint translates a multi-constraints problem into a single-constraint problem. Let $\mu = \{\mu_1, \ldots, \mu_m\}$ be a non-negative vector, the surrogate relaxation of 0/1 MKP is given by:

$$\begin{cases} Max \sum_{i=1}^{n} p_i x_i & (2.7) \\ s.t \sum_{i=1}^{n} \left(\sum_{j=1}^{m} \mu_j w_{ij} \right) x_i \leq \left(\sum_{j=1}^{m} \mu_j \right) C_j \ ; & (2.8) \\ x_i \in \{0,1\} \ ; \quad \forall i = 1, \ldots, n; & (2.9) \end{cases}$$

(Pirkul, 1987) has developed a heuristic that makes use of surrogate duality. A feasible solution was obtained by packing items into the knapsack in decreasing order of their ratios defined by $p_i / \sum_{j=1}^{m} \mu_j w_{ij}$ (for $i \in 1, \ldots, n$), where μ_j is the surrogate multiplier for constraint j ($\in \{1, \ldots, m\}$) that is calculated by using a descent procedure. Computational results proved the effectiveness of the proposed heuristic, regardless of instances size, against other heuristics developed by (Loulou & Michaelides, 1979) and (Balas & Martin, 1980). Besides, (Fréville & Plateau, 1996) have presented a heuristic for the bidimensional knapsack problem employing a reduction procedure, a bound based upon surrogate relaxation, and partial enumeration. The performance of their heuristic was evaluated over randomly generated problems with up to 750 items and compared with an exact algorithm of (Gavish & Pirkul, 1985). On the other side, (Chu & Beasley, 1998) have incorporated in the standard genetic algorithm a surrogate relaxation based-heuristic, which applies an alternative method for finding surrogate multipliers that involves solving the LP relaxation of the original problem and using the dual variables from that solution as surrogate multipliers.

Additionally, (Montaña, et al., 2008) have suggested a new method for computing surrogate multipliers proving its effectiveness against that proposed by (Chu & Beasley, 1998) as well as other methods based on LP relaxations. Other heuristics based on surrogate relaxation are given by (Boyer, et al., 2009). Their first developed heuristic consists of solving the surrogate relaxed problem via an improved dynamic-programming algorithm, while the second utilizes, additionally to dynamic programming algorithm, a limited Branch and Cut procedure. The simulations were presented on randomly generated instances and small-to-large standard benchmark instances. Both approaches yielded better results with a smaller gap between the best solution and optimal one, in comparison with the heuristics proposed by (Freville & Plateau, 1994), (Bertsimas & Demir, 2002), and (Boyer, 2004).

A combination of Lagrangian relaxation with surrogate relaxation gives rise to another type, called composite relaxation (Greenberg & Pierskalla, 1970) and given by:

Let $Q \subseteq \{1,\dots m\}$, $q = |Q| \neq 0$, $\lambda \in \mathbb{R}_+^q$, $\mu \in \mathbb{R}_+^m$

$$\begin{cases} Max \sum_{i=1}^{n}(p_i - \sum_{j\in Q}\lambda_j w_{ij})x_i + \sum_{j\in Q}\lambda_j C_j & (2.10) \\ s.t \sum_{i=1}^{n}\left(\sum_{j=1}^{m}\mu_j w_{ij}\right)x_i \leq \left(\sum_{j=1}^{m}\mu_j\right)C_j \ ; & (2.11) \\ x_i \in \{0,1\} \ ; \quad \forall i = 1,\dots,n; & (2.12) \end{cases}$$

It is clearly seen that the Lagrangian and surrogate relaxations are two variants of composite relaxation. To the best of our knowledge, no heuristic based on this kind of relaxation is published until now in 0/1 MKP literature. So, it can be a good perspective for future work.

4 Metaheuristics

The metaheuristics dedicated to solving 0/1 MKP can be classified into two categories: Single-solution metaheuristics that consist of modifying and improving a single candidate solution, and population based-metaheuristics wherein multiple candidate solutions are maintained and improved through several operations.

4.1 Single-solution metaheuristics

4.1.1 Tabu search

The Tabu Search (TS) is a metaheuristic introduced by (Glover, 1990). It can be seen as simply a combination of local search procedure with short-term memories. Two first basic elements of any TS algorithm are the definition of its *search space* and its *neighborhood structure*. The exploitation of the neighborhood allows moving from the current solution to its best neighbor that is not necessarily better than it. To prevent cycling between a local optimum and its best neighbor, a short-term memory (called tabu list) is introduced containing the last visited solutions and updated by removing from the list the most visited solution at each iteration. The tabu list length and stopping criterion are two parameters to adjust in the TS algorithm.

(Dammeyer & Voss., 1993) have presented a TS algorithm for solving 0/1 MKP based on reverse elimination. The proposed TS yielded 41 optimal solutions of 57 test problems. (Battiti & Tecchiolli, 1995) have proposed TS algorithm that makes use of a tabu list with variable size. Computational results were carried out on problems having up to 500 constraints and 500 items. (Glover & Kochenberger, 1996) have suggested a TS characterized by a recency-based and frequency-based memory information. Their method was enhanced by a strategic oscillation

scheme that navigates both sides of the feasibility boundary (side of infeasible solutions and side of feasible ones). The experiments proved the efficiency of their proposed TS for 57 standard instances. (Løkketangen & Glover, 1996) have developed a probabilistic version of TS in which removing a potential solution is controlled by a probabilistic process. The proposed TS yielded 13 optimal solutions of 18 test problems containing up to 30 dimensions and 90 items. (Niar & Freville, 1997) have proposed a parallel TS that uses a dynamic manner of setting TS parameters, and makes an equilibrium between intensification and diversification strategy through a parallel cooperative search. ((Vasquez & Hao, 2001) and (Vasquez & Vimont, 2005)) have proposed an efficient algorithm that combines TS with linear programming. The simulations showed that their approach performs well for 0/1 MKP instances issued from OR-Library[14] against other state-of-art algorithms.

4.1.2 Variable Neighborhood Search

The Variable neighborhood search (VNS) was introduced by (Hansen & Mladenović, 1997). It consists of systematic changes in neighborhood structures during the search process. It is based on the following facts: (a) A local optimum relative to one neighborhood structure is not necessarily a local optimum for another one. (b) A global optimum is a local optimum of all neighborhood structures. (c) For many problems, all or a large majority of the local optima are relatively close to each other.

(Hanafi, et al., 2009) have proposed an approach that incorporates dynamic refining of the lower and upper bounds within the Variable Neighborhood Decomposition Search (*i.e.* a two-level VNS scheme based on the decomposition of the problem at hand). In their approach, the set of neighborhoods is generated by exploiting information obtained from a series of LP relaxations. In each iteration, new constraints are added to the current sub-problem and both sub-problem and its linear relaxation are solved. The proposed VNS-based approach achieved comparable results against the state-of-art 0/1 MKP solving methods for 11 benchmark instances ranging from 100 to 2500 items and 15 to 100 constraints. Besides, a variant of VNS called Relaxation Guided Variable Neighborhood Search is developed by ((Puchinger & Raidl, 2005) and (Puchinger & Raidl, 2008)). To evaluate the performance of their approach, they have used 0/1 MKP as a test problem. Recently, (Tasgetiren, et al., 2015) have combined VNS with a differential evolution algorithm (Neri & Tirronen, 2010) in order to solve 0/1 MKP. The hybrid approach was tested upon benchmark instances from the OR-Library. Empirical results showed its efficiency and its superiority to the best-performing algorithms from 0/1 MKP literature.

4.1.3 Simulated annealing

Simulated Annealing (SA) was introduced by (Kirkpatrick, et al., 1983). It is one of the first metaheuristics inspired from the physical phenomena happening in the solidification of

[14] is a collection of test data sets for a variety of Operations Research (OR) problems.

fluids such as metals. By analogy with the physical process, the energy E of a system (that should be minimized) corresponds to the objective function of the problem at hand and the temperature T is regarded as the stopping criterion of the algorithm. In fact, at each virtual annealing temperature, the SA algorithm generates a new potential solution (a neighbor of the current solution) of the problem by altering the current solution that will then be either accepted or rejected. The acceptance or rejection of the modified solution is based firstly on the variation of energy ΔE: If $\Delta E < 0$, then the generated solution is accepted. If not, the generated solution is accepted according to Metropolis criterion (Metropolis, et al., 1953) as following:

$$\text{Let } \varepsilon \in [0,1]: \quad \begin{cases} accepted\ solution & if\ e^{-\Delta E/T} \leq \epsilon \\ rejected\ solution & otherwise \end{cases} \tag{2.13}$$

(Drexl, 1988) has applied SA algorithm to solve 0/1 MKP. Computational results were presented for 57 standard test problems from the literature indicating that the algorithm converges rapidly towards the optimum solution. Besides, (Leung, et al., 2012) have presented a hybrid SA method. Firstly, a constructive method based on a fitness strategy is used to generate solutions and another one is applied to search for better solutions. Then, SA is applied to improve the quality of obtained solutions. This method is addressed to deal with rectangle knapsack problem with two-dimensions in which the items to be packed into the knapsack have all a rectangular-shaped form. The outcome results upon 221 test instances showed that the proposed method performs better than some previous existing algorithms. Recently, (Gupta & Arora, 2015) have proposed a new SA algorithm that incorporates effective fitness landscape parameters. Their basic idea is to ignore the association between the search space and the fitness space while focusing only on the comparison between the current solution and the optimal solution. Their algorithm provided a good tradeoff between computational time and solution accuracy when compared with three evolutionary algorithms.

4.2 Population-based metaheuristics

The based-population metaheuristics have attracted much attention from many researchers to solve 0/1 MKP (see the work of (Laabadi, et al., 2018a)), in particular Genetic Algorithms, followed by Particle Swarm Optimization and Ant Colony Optimization algorithms that are clustered in one group labeled Swarm Intelligent algorithms.

4.2.1 Genetic algorithms

The Genetic algorithm (GA) was introduced by (Holland, 1975). It is inspired by Charles Darwin's theory of natural evolution (Darwin, 1913). In a GA, the solutions evolve over many generations according to the principles of natural reproduction that are selection, crossover, and mutation. More details will be described in the next chapter.

(Khuri, et al., 1994) have solved 0/1 MKP by GA using a penalty function to deal with infeasible solutions that are allowed to participate in the search process. Their GA is tested on a small

number of standard test problems ranging from 15 to 105 items and 2 to 30 constraints, indicating that the results found are modest. On the other hand, in the GA of (Hoff, et al., 1996), only feasible solutions were allowed. Their proposed GA has successfully obtained optimal solutions for 56 out of 57 standard test problems taken from the literature. (Thiel & Voss, 1994) have combined GA with TS, however, the obtained results were not competitive in solving the standard instances of 0/1 MKP. (Chu & Beasley, 1998) have developed one of the first successful GA to solve 0/1 MKP. Their approach differs from previous GAs in the way that a problem-specific knowledge repair operator is incorporated into the core of GA in order to convert infeasible solutions to feasible ones. Additionally, they have proposed new test instances of 0/1 MKP that are characterized by different sizes and various degrees of complexity (available in OR-Library site). The simulations presented on these test instances showed that their proposed GA yields high-quality solutions by requiring a modest runtime in comparison with heuristics of (Magazine & Oguz, 1984), (Pirkul, 1987), and (Volgenant & Zoon, 1990). Recently, (Martins, et al., 2014) have analyzed the usefulness of different versions of the Linkage Tree Genetic Algorithm (LTGA) by using Chu and Beasley's GA as reference. The experiments showed that Chu and Beasley's GA outperforms LTGA versions over their proposed test instances. In contrast, (Lai, et al., 2014) have proposed a GA that performs well than Chu and Beasley's GA over the same test instances, in which they have combined two types of information: about the LP relaxation of (Raidl, 1998) and the pseudo-utility ratio of (Chu & Beasley, 1998). On the other hand, (Varnamkhasti & Lee, 2012a) and (Varnamkhasti & Lee, 2012b) have been interested in sexual GA in which only the couples of chromosomes with opposite sex take place in crossover operation. Their proposed GAs were compared favorably with other heuristic approaches. Moreover, (Yang, et al., 2013) have integrated attribute reduction in Rough Set [15] into a crossover operator. The authors applied RS to formulate a set of items and used the attribute reduction process to eliminate redundant items. The experiments tested on 0/1 MKP benchmark data have demonstrated that the methodology adopted is a good alternative to improve the performance of GA.

4.2.2 Swarm intelligent algorithms

Ant colony optimization algorithm. Ant colony optimization (ACO) algorithm was introduced by (Dorigo, et al., 1996). It is based on the ant's capability of finding the shortest path from the nest to a food source. Each ant traces a path (i, j) by laying an amount of pheromone τ_{ij}. Each path is represented by an arc (i, j) of the graph $G = (N, A)$. The intensity of the pheromone in an arc is an indicator of the utility of this latter to build better solutions. Initially, a constant amount of pheromone $(= 1)$ is allocated to all arcs. At each iteration, an ant k sitting at the i-th node of the graph chooses the node j with a probability $p_{ij}(k)$ given by:

[15] After probability theory and fuzzy set theory, the rough set theory is an important mathematical tool to deal with imprecise, inconsistent, and incomplete information (see (Zhang, et al., 2016))

Let N_i^k be the neighborhood of ant k when sitting at the i-th node,

$$p_{ij}(k) = \begin{cases} \frac{\tau_{ij}^{\alpha}}{\sum_{l \in N_i^k} \tau_{ij}^{\alpha}} & if\ j \in N_i^k \\ 0 & otherwise \end{cases} \quad (2.14)$$

The pheromone level is updated at each iteration by:

$$\tau_{ij}(k+1) = \rho \tau_{ij}(k) + \Delta\tau_{ij}(k) \quad (2.15)$$

Where $\Delta\tau_{ij}(k)$ is the performance of an ant k and $1-\rho$ is the evaporation ratio, where $0 < \rho < 1$.

(Leguizamon & Michalewicz, 1999) have developed the first ant colony optimization algorithm for solving 0/1 MKP. Their proposed ACO approach works better than an evolutionary algorithm on 20 out of 31 test instances taken from OR-Library. (Fidanova, 2002) has used a particular implementation of ACO, namely the ant colony system (ACS), which incorporates separately three types of heuristics. The empirical results proved the effectiveness of its proposed algorithm for 30 benchmark instances having 5 constraints and 100 items. Additionally, (Ke, et al., 2010) have proposed an ACO approach considered as an extension of Max-Min Ant System. They have also proposed a local search procedure to improve solutions constructed by ants. The results showed that their proposed algorithm competes efficiently with other promising approaches to solve the 0/1 MKP. Besides, (Kong, et al., 2008) have proposed a Binary Ant System (BAS) to deal with the 0/1 MKP. This algorithm uses a pheromone laying method specially designed for the binary solution structure and applies a repair operator to restore the feasibility of solutions. The computational results demonstrated the effectiveness of BAS among other ACO-based approaches. ((Liu & Lv, 2013) and (Fingler, et al., 2014)) have proposed two different implementations of a parallel ACO to tackle with 0/1 MKP. Recently, (Nakbi, et al., 2015) have proposed a hybrid approach for solving 0/1 MKP that combines ACO with Lagrangian relaxation based-heuristic. Experiments on large benchmark instances proved the superiority of the hybrid approach against the ACO algorithm and the Lagrangian heuristic when they are tested separately.

Particle swarm optimization algorithms. The particle swarm optimization (PSO) algorithm is introduced by (Kennedy & Eberhart, 1995). It is inspired by the social behavior and dynamics of insects, birds, and fishes. In analogy to birds, the PSO algorithm proceeds as following. A number of agents (called particles) constitute a swarm moving around the search space looking for the best solution. Each particle has a position defined in d-dimensional space. At each iteration of the algorithm, the particles adjust their positions according to an imaginary velocity, the personal best positions, and a global best position in the swarm. More details will be described in chapter 6.

(Wang, et al., 2009) have introduced a binary version of the PSO algorithm, in which they have incorporated a mutation operator and a dissipation operator to keep the diversity of swarm and enhance the search ability. The simulations were carried out on a small number of 0/1 MKP instances, indicating that the proposed algorithm outperforms two other PSO-based approaches. (Langeveld & Engelbrecht, 2012) have developed another binary implementation of PSO algorithm, called set-based PSO algorithm. The computational results were presented on a large number of test instances taken from the 0/1 MKP literature, revealing that the proposed algorithm performs better than three other PSO-based approaches. Moreover, (Chih, et al., 2014) have proposed two binary versions of PSO algorithm to solve 0/1 MKP. A binary PSO with time-varying acceleration coefficients and a chaotic binary PSO with time-varying acceleration coefficients. The two proposed algorithms were tested upon 116 benchmark problems from OR-Library. The results showed that both algorithms outperform two other existing PSO-algorithms. Similarly, (Lin, et al., 2016) have proposed a binary PSO based on the surrogate information with proportional acceleration coefficients. Their proposed PSO approach was tested using 135 benchmark problems from the OR-Library in order to prove its effectiveness in solving 0/1 MKP. On the other side, (Bonyadi & Michalewicz, 2012) have introduced a new encoding scheme, a new initialization phase, and an improvement procedure. The simulations showed that the resulting PSO approach is faster than most of the other comparative methods for several 0/1 MKP benchmark instances. Recently, (Chih, 2015) have proposed a self-adaptive repair operator that uses more than one pseudo-utility ratio and dynamically changes them as the PSO algorithm runs. The proposed repair operator was then incorporated in the PSO-based approach for solving 0/1 MKP. The experiments showed that the proposed PSO algorithm could compete efficiently with other existing PSO-based approaches as well as other population-based algorithms.

Finally, developing population-based metaheuristics for 0/1 MKP never ends; this is due to its miscellaneous applications in real life. Table 1 lists some of the latest metaheuristics for solving 0/1 MKP.

Table 1. List of recent population-based algorithms dealing with 0/1 MKP

References	Algorithms
(Ji, et al., 2013)	Artificial Bee Colony Algorithm Merged with Pheromone Communication Mechanism
(Sabba & Chikhi, 2014)	Discrete Binary Version of Bat Algorithm
(Gong, et al., 2011)	Hybrid Artificial Glowworm Swarm Optimization
(Azad, et al., 2014)	Improved Binary Artificial Fish Swarm Algorithm
(Meng, et al., 2017)	An Improved Migrating Birds Optimization
(Zhang, et al., 2016)	Binary Artificial Algae Algorithm
(Baykasoğlu & Ozsoydan, 2014)	An Improved Firefly Algorithm
(Kong, et al., 2015)	New Binary Harmony Search Algorithm
(Liu, et al., 2016)	Binary Differential Search Algorithm
(Wang, et al., 2015)	Human Learning Optimization Algorithm
(Hanafi & Wilbaut, 2008)	Scatter Search

(Yuan & Yang, 2016)	Weight-Coded Evolutionary Algorithm
(Deane & Agarwal, 2012)	Augmented Neural Networks Algorithm
(Fingler, et al., 2014)	Ant colony optimization with CUDA
(Bolaji, et al., 2018)	Binary Pigeon-Inspired Algorithm
(Abdel-Basset, et al., 2018)	Modified flower pollination algorithm

5 Conclusion

This chapter is devoted to resolution methods of 0/1 MKP. We presented a state-of-art including the most recent works. We provided an overview of the exact methods as well as the heuristic approaches. Thus, we have remarked that a few articles have been interested in solving 0/1 MKP with exact methods, whilst the heuristic approaches are frequently encountered either the heuristics or the metaheuristics. In the next chapter we will expose our first proposed algorithm to deal with the 0/1 MKP.

Chapter 3: An improved sexual genetic algorithm for solving 0/1 MKP

1 Introduction

During the 1960s and 1970s, as soon as computing systems of more credible power came into existence, many attempts are undertaken to model the process of evolution and use it as an optimization tool for engineering problems. Among the early approaches based on an evolutionary process are evolution strategies (Rechenberg, 1965), evolutionary programming (Fogel, et al., 1966), and genetic algorithms. Thereafter, the genetic algorithm became the widely held and prominent algorithm after the publication of the book "*Genetic Algorithms in Search, Optimization and Machine Learning*" by (Goldberg, 1989). The aforementioned approaches including the genetic algorithm are based upon Darwin's mechanism of evolution. In fact, according to (Darwin, 1913), the mechanism of evolution rests on a competition that aims to select the most well adapted individuals to their environment, and transmit the useful characteristics to their children through a form of cooperation implemented by the sexual reproduction. By this way, the children allow the survival of fittest parents (*i.e.* more adapted individuals).

The general principles of GAs including selection, crossover, mutation, and replacement strategies were first discussed by (Holland, 1975), and then applied in various optimization problems. Due to the shortcoming of GA consisting of its premature convergence, considerable research waxed and waned from year to year to improve its performance that largely depends on the selection and crossover operations. The selection operation is a critical step because if the individuals has been selected in an intelligent manner then it will increase the genetic diversity, thereby avoiding premature convergence. However, incorporating an appropriate crossover operator is very important because it can exploit more the search space.

In the present chapter, we propose a selection strategy inspired from that of (Varnamkhasti & Lee, 2012a) as well as two variants of a crossover operator developed by (Aghezzaf & Naimi, 2009). These genetic operators are combined in the same framework giving rise to an efficient GA. This chapter is organized as follows. We begin by describing the principles of the standard genetic algorithm. We then describe the proposed selection strategy as well as the crossover operators in sections 3 and 4. Section 5 is devoted to experimental results. Finally, section 6 concludes the chapter and indicates possible directions for future work.

2 Traditional genetic algorithms

Inspired by evolutionary biology, the genetic algorithm (GA) belongs to the huge category of bio-inspired algorithms (see Figure 4). It operates on a set of randomly generated individuals, evaluates the individuals to determine which one can survive, then iteratively applies selection, crossover, and mutation operators to reproduce new individuals until satisfying a stopping criterion. GA is found as a powerful tool for solving hard and complex

optimization problems. This success mainly depends on the genetic operators used, in particular, selection and crossover operators (see hereinafter).

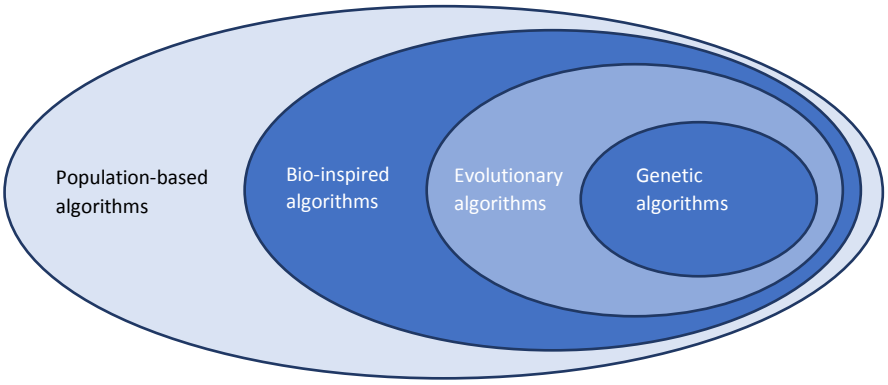


Figure 4. Classification of the genetic algorithm

In the vocabulary of the evolutionary algorithms, the individuals subjected to evolution are designed by "chromosomes", which correspond to the candidate solutions of the problem at hand. Each chromosome is composed of "genes" that correspond to decision variables of the optimization problem. The set of the chromosomes treated simultaneously by the evolutionary algorithm constitutes a "population". This latter evolves during a succession of iterations, called "generations", until a termination criterion while taking into account the quality of generated chromosomes that is measured thanks to the so-called "fitness function". During each generation, a succession of operators is applied to chromosomes to generate a new population for the next generation. The chromosomes subjected to genetic operators are called "parents", while the chromosomes originating from the application of genetic operators are called "offspring chromosomes". Then, the offspring chromosomes become parents in the next generation. Figure 5 shows the main steps of a standard GA, and then the following sub-sections describe it in detail.

- Generate randomly an initial population of chromosomes
- Repeat until a stopping criterion is met:
  - Execute selection operation in the whole population.
  - Recombine pairs of parents to breed offspring chromosomes by applying a crossover operator.
  - Mutate some of the offspring chromosomes with a certain probability
  - Replace the population of parents with their children in the next generation
- Return the best chromosome as a global optimum of the considered problem.

Figure 5. Main steps of a standard GA

2.1 Representation of chromosomes

One of the most important decisions to make before implementing a GA is deciding the representation of potential solutions for a given problem. The representation of chromosomes

is highly dependent on the nature of the problem. Hence, defining non-convenient representation may deteriorate the algorithm performance. The chromosome is usually represented by a string of binary variables such as the case of 0/1 MKP (see the next section) or by real numbers like in berth allocation problem (Arango, et al., 2013). However, an integer representation scheme is used for example in optimization function with integer variables, in addition to a permutation representation scheme that is based on the order of events (*e.g.* Job-Shop Scheduling problem (Driss, et al., 2015)) or based on adjacencies (*e.g.* Traveling Salesman Problem (Király & Abonyi, 2010)). The length of a chromosome is determined by referring to problem instances

2.2 Selection strategy

The selection strategy is a main stage of GA that consists of selecting parents from the entire population and placing them in a mating pool[16] for later breeding. The selection operation is applied according to a certain preference criterion that could play an important role in controlling the level of population diversity. The classical selection strategies tend to use fitness-based criterion, these included:

Tournament Selection: It selects *t* chromosomes at random from the population and picks out the fittest one (*i.e.* having the highest fitness value) to become a parent. A simple way is setting t to 2, in such case this selection operator is named binary tournament selection.

Roulette Wheel Selection: It exploits the metaphor of a biased roulette game, where the roulette wheel compartments correspond to chromosomes and the size of each compartment is proportional to the fitness of each chromosome. The selection of a chromosome is guided by a fixed point chosen at random on the wheel, then the wheel is rotated and the chromosome that comes in front of the fixed point is chosen as a parent. Its probability of being selected is:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \quad ; \qquad \forall i = 1, \dots, N \tag{3.1}$$

Where N is the number of chromosomes in the population (or population size), and f_i is the fitness value of a chromosome i in the population.

Statistical Universal Selection: It considers a straight-line segment partitioned in as many zones as chromosomes in the population. The size of each zone is proportional to the fitness value of each chromosome. The selection of chromosomes is designated by a set of equidistant pointers placed over the line as many as there are chromosomes to be selected. Let Nb be the number of chromosomes to be selected, the distance between the pointers is thus $1/Nb$, and the position of the first pointer is randomly generated between 0 and $1/Nb$.

[16] The mating pool is a concept used in evolutionary computation to indicate the place where the selected chromosomes are grouped for later breeding.

Truncation Selection: It simply retains a proportion of fittest chromosomes from the population according to a predefined truncation threshold, which is a preset parameter ranging between 10% and 50%.

2.3 Crossover operator

In the crossover stage, the parents placed in the mating pool are crossed to reproduce new chromosomes. During this process, the parents share the information between them, thereby breeding new offspring chromosomes that inherit some features of their ancestors. The crossover operator is considered as a primary search operator according to (Lin & Yao, 1997) since it exploits the available information of the population and transfers it through generations. Different forms of crossing can be used:

The single-parent crossover technique: It involves some modifications applied to a parent chromosome, which results in a new progeny becoming the offspring chromosome.

The two-parent crossover technique: It is a reproduction technique that combines two parents to generate offspring individuals. The commonly used two-parent crossover techniques include:

- ✓ One-point crossover operator, in which both parents are split into a left genome[17] and right genome, where the left genome or the right genome of each parent are the same length. Then each offspring gets the left genome of one parent and the right genome of the other parent. The split position is called a crossover point.

- ✓ The two-point crossover operator is a general case of the one-crossover operator, in which two crossover points are considered.

- ✓ The uniform crossover operator exchanges the genes rather than the parents' genomes. It is considered as a multipoint crossover operator. Practically, a "template string" is used that consists of a binary string having the same length as the parents. The value "0" at the *i*-th locus of the template leaves the value of the *i*-th gene of both parents unchanged, and the value "1" activates an exchange of the value of corresponding genes. The template is generated at random for each pair of parents.

These three crossover operators are explained by illustrative examples in Figure 6. Note that the symbol "|" represents a randomly selected point.

The multi-parent crossover technique: In this method, k ($k > 2$) parents are chosen from the population and combined to breed one or more offspring individuals. For instance, in the case of $k = 3$, three parents are randomly chosen. Each gene of the first parent is compared with

[17] A genome is a set of genes

the gene having the same locus in the second parent. If the genes have the same value, then one of both genes is inherited by the offspring. Otherwise, the gene of the third parent is selected.

One-point crossover operator	Parent 1: {11001|011} Parent 2: {11011|111}	Offspring 1: {11001111} Offspring 2: {11011011}
Two-point crossover operator	Parent 1: {110|010|11} Parent 2: {110|111|11}	Offspring 1: {11011111} Offspring 2: {11001011}
Uniform crossover	Random template:{11000101} Parent 1: {11001011} Parent 2: {11011111}	Offspring 1: {11001111} Offspring 2: {11011011}

Figure 6. Examples of two-parent crossover techniques by using binary representation

The crossover operation can be applied in selected parents from the population with high-level probability ranging between 0.50 and 0.95.

2.4 Mutation strategy

While crossover enhances the intensification in the search space, the mutation extends the domain of search to restore lost or unexplored individuals in the population, thereby increasing the diversity in the population of GA. When an individual is chosen for mutation, a random choice is made of some of its genes and then this latter are modified. In other words, the mutation operator has the effect of destroying the structure of a potential solution for producing a new one with a different and likely promising structure. The classical mutation operators are applied depending on the encoding scheme of chromosomes; below you will find some examples of classical mutation strategies used in the case of binary representations:

Bit-flip Mutation: It consists of selecting at random one gene, and then changing its value from 0 to 1 or from 1 to 0.

Interchanging Mutation: Two random genes of the offspring are chosen and their corresponding values are interchanged.

Reversing Mutation: A random position is chosen and the gene value next to that position is reversed. Figure 7 gives some illustrative examples.

Bit flip mutation	Offspring : {11001111}	Mutated offspring : {01001111}
Interchanging mutation	Offspring : {11001111}	Mutated offspring : {11101101}
Reversing mutation	Offspring : {11001111}	Mutated offspring : {11001101}

Figure 7. Examples of mutation strategies useful for a binary representation

The mutation operation can be applied in offspring chromosomes with extremely low probability ranging between 0.001 and 0.01.

2.5 Replacement strategy

Once the offspring chromosomes undergo these above-mentioned operators, they are integrated into the population of the next generation according to a replacement strategy. This latter defines how to select the next generation members while preserving diversity through generations. The commonly used replacement strategies in the literature are:

Generational strategy, in which all generated offspring chromosomes replace their parents to form the population for the next generation

Steady-state replacement, in which only some offspring chromosomes replace their parents to form the population for the next generation.

Elitist strategy, that consists of selecting the fittest chromosomes from the offspring chromosomes as well as their parents, in order to form a fittest population for the next generation.

3 The proposed sexual genetic algorithm

3.1 Encoding schema

As the proposed genetic algorithm is designed to deal with 0/1 MKP, each chromosome corresponds to a set of items to be placed in the knapsack. Considering the structure of 0/1 MKP, a chromosome k is represented by a binary string $X_k = \{x_{k1}, x_{k2}, \ldots, x_{kn}\} \in \{0,1\}^n$, where n is the total number of items to be packed into the knapsack and x_{ki} is a gene of the chromosome k wherein the genetic information is stored. This information is coded by 1 if an item is inserted in the knapsack and 0 if the item is rejected out of the knapsack. Figure 8 shows a typical representation of a chromosome having ($n = 7$) candidate items.

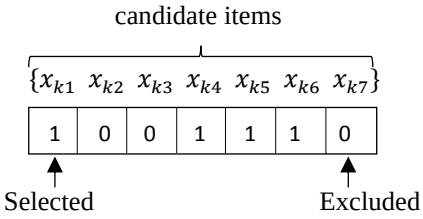


Figure 8. Binary encoding schema

The search space is initially represented by a population randomly generated by binary values: 0's and 1's. However, random allocation of genes into chromosomes will end up with some infeasible solutions in the initial population, because the knapsack constraints may not all be satisfied. In the proposed algorithm, we apply a repair operator to retain the feasibility of chromosomes.

3.2 Repair operator

There are several ways for dealing with infeasible solutions in GA: Using a representation ensuring that all solutions are feasible (Hoff, et al., 1996), applying a penalty function to penalize the infeasible solutions (Khuri, et al., 1994), or applying a repair operator that converts infeasible solutions into feasible ones (Chu & Beasley, 1998). Lately, the technique of repair operator becomes widely used in GAs. The repair mechanism is based upon a greedy heuristic to repair and improve infeasible solutions. The general technique used to design such greedy heuristics for 0/1 MKP follows the notion of pseudo-utility ratio (denoted u_i for an item i) that is the key factor of their success. Thus, several ways to calculate the pseudo-utility ratio have been proposed in 0/1 MKP literature including:

- The surrogate relaxation-based ratio of (Pirkul, 1987):

$$u_i = \frac{p_i}{\sum_{j=1}^{m} r_j\, w_{ij}},\ \ \forall i = 1,\ \ldots,\ n \tag{3.2}$$

Where $r = (r_1, r_j, \ldots, r_m)$ is a set of surrogate multipliers.

- The relative mean resource occupation proposed by (Kong, et al., 2015):

$$u_i = \frac{p_i}{\sum_{j=1}^{m} \frac{w_{ij}}{m\, c_j}},\ \ \forall i = 1,\ \ldots,\ n \tag{3.3}$$

- The profit density used by (Zouache, et al., 2016) :

$$u_i = \frac{p_i}{\sum_{j=1}^{m} w_{ij}}, \forall i = 1,\ \ldots,\ n \tag{3.4}$$

In the proposed GA, we will use the ratio represented by equation (3.4) since it is simple to implement and faster than the surrogate relaxation-based ratio, and has fewer parameters than the relative mean resource occupation. Indeed, the repair operator using the profit density ratio is performed in two phases: A "DROP" phase and an "ADD" phase. In the DROP phase, the

items having the lowest profit density are removed iteratively until all capacity constraints of the knapsack are satisfied. Then, in the ADD phase, the items having the highest profit density are added in turn until at least one capacity constraint is violated.

3.3 The proposed sexual selection

The selection operators such as binary tournament selection, roulette wheel selection, or others based on fitness level may gradually lose the diversification in the population by choosing only fittest chromosomes, and hence the GA can trap on premature convergence. An efficient implementation of the selection strategy called "sexual selection", which takes into account the gender of chromosomes, is developed to maintain the diversity in the population. (Miller, 1994) and (Miller & Todd, 1995) have shown, in their studies of selection mechanisms, that the sexual selection strategy may enhance the diversification while exploring more the search space. It can help populations to escape from local optima by using a stochastic process and then facilitates the emergence of evolutionary innovations. The authors have considered the sexual selection process as a large-scale evolution that promotes the discovery and propagation of new chromosomes over many generations, while the mutation is viewed as a small-scale evolution strategy. For this reason, we have focused our attention on this type of selection.

In that sense, (Lis & Eiben, 1996) have developed a new selection strategy inspired by the sexual selection principle. They have applied this strategy to multi-criteria optimization problems in which each criterion consists of a specific sex. Then, a multi-parent crossover operator is applied to generate offspring chromosomes with different genders. (Goh, et al., 2003) have proposed a sexual selection for solving some well-known engineering problems. In their method, they have divided the population into male and female chromosomes in a random way. Then, they have applied the principle of female choice to build couples. All the female chromosomes are getting to be crossed, whilst their mates are selected through tournament selection. Otherwise, (Varnamkhasti & Lee, 2012a) have developed a new technique to build couples based on three preference criteria that are Hamming distance between two chromosomes, the number of active genes of each chromosome, or the fitness value. Very recently, (Laabadi, et al., 2018b) have used a crossover operator that seems synergetic with the sexual selection proposed by (Varnamkhasti & Lee, 2012a) to improve the performance of GA. Such a combination is tested on randomly generated instances of 0/1 unidimensional knapsack problem.

Our proposed operator is inspired by the technique of (Varnamkhasti & Lee, 2012a) in which the authors divide the initial population into two groups having opposite sex: Group of male chromosomes and group of female chromosomes. They can distinguish the sex of a chromosome through its label: A chromosome k is a female if k is an odd number and it is a male if k is even. In their technique, the authors select female chromosomes from the female group by tournament selection with tournament size t. Based on a female choice policy, each

selected female chromosome picks out t males at random from the male group in order to choose its mate. The females work sequentially without replacement, which means that each female will only get to participate in mating once. However, the selection of male chromosomes is based on a traditional similarity-based criterion that consists of measuring the distance between bit strings by means of Hamming distance, or with a fitness-based criterion, or by focusing on the value of genes. The pseudo-code of their sexual selection strategy is detailed in Algorithm 2.

13. **Input: Population of k feasible chromosomes, $k = 1, \dots, N$**
14. For $k = 1: N$
15. If k mod $2 == 0$, then the chromosome k is a male, else the chromosome k is a female
16. Select females to be crossed by tournament selection with tournament size t
17. For each selected female
18. Select randomly t males
19. For $j = 1: t$ males
20. Calculate the HD_j = Hamming distance (X_{female} , X_j)
21. Select the male having a maximum Hamming distance
22. If $card\{\underset{j}{max}\, HD_j\} \geq 2$, then calculate the fitness value $f(X_j) = \sum_{i=1}^{n} p_i\, x_{ji}$
23. Select the male with the highest fitness value $\underset{j}{max}\{f(X_j)\}$
24. If $card\{\underset{j}{max}\{f(X_j)\}\} \geq 2$, then calculate the occurrence of nonzero genes (active genes) $AG_j = \underset{i=\,1,\dots,n}{card}\{x_{ji} \,/\, x_{ji} \neq 0\}$
25. Select the male with the highest number of active genes
26. If $card\{\underset{j}{max}\, AG_j\} \geq 2$, then choose the male randomly
27. End for
28. End for
29. End for
30. **Output: Pairs of chromosomes with opposite sex**

Algorithm 2. The pseudo code of the sexual selection strategy

We modified the above selection strategy in such a way that the structure of the problem is taking into consideration, giving rise to a new variant working in phenotype space[18] rather than genotype space. The phenotype space makes use of real values to represent genes, while the genotype space uses a binary representation. In order to adapt the representation of a chromosome to phenotypic space, we attribute to each item *i* a fitness value given by:

$$f_i = \frac{p_i}{\sum_{j=1}^{m} w_{ij}}, \quad i = 1, \dots, n \tag{3.5}$$

Hence, the value of the gene x_{ki} of the chromosome *k* takes the fitness value f_i if the item *i* is selected and 0 otherwise. Similarly to the methodology of (Varnamkhasti & Lee, 2012a), our proposed selection strategy rests on the principle of female choice and incorporates the same selection schema of female chromosomes. However, the mate of each female chromosome is selected based on other seduction functions. Firstly, the unmated female chooses the male phenotypically dissimilar from itself. Then, when the candidate males have the same degree of

[18] In genetics, the *phenotype* is the physical and behavioral traits of the organism, for example, size and shape, metabolic activities, and patterns of movement. In evolutionary computation, the phenotype space corresponds to the space of decoded solutions.

similarity, the female chooses the fittest one. Then again, when the candidate males have the same fitness level, the female chooses the male having lightweight items (for giving chance to other items to be packed into knapsack in upcoming genetic operations). Finally, when the candidate males have the same weight of items, the female picks out its mate in a random way. In fact, by applying these four criteria alternatively, the proposed algorithm may ensure a balance between intensification and diversification through the evolution process of GA. The pseudo-code of the modified sexual selection strategy is presented in Algorithm 3.

To measure similarity within the genotypic space, Hamming distance is frequently used. Nevertheless, several distance functions are employed when the similarity between individuals is measured within the phenotypic space, such as L_p norm that is considered as a natural extension of its traditional use in topological space. According to the value of p, it leads to different distances: Manhattan distance (L_1 norm), Euclidian distance (L_2 norm), and Minkowski distance ($L_{p\geq 1}$ norm). Generally, the L_p norm (or Minkowski distance) is defined by the following equation:

$$x, y \in \mathbb{R}^d, p \in \mathbb{Z}, L_p(x,y) = \sum_{i=1}^{d} (\|x_i - y_i\|^p)^{1/p} \tag{3.6}$$

(Aggarwal, et al., 2001) showed that the L_p norm depends heavily on the value of p. It degrades rapidly with a higher value of p, more especially, in the problem with a high dimensionality d. Consequently, it is preferable to use lower values of p. Thus, L_1 norm is the most preferable, followed by the L_2 norm, and so on. So, for this reason we choose the Manhattan distance to deal with disassortative mating[19].

---

1. **Input: Population of k feasible chromosomes, $k = 1, \dots, N$**
2. For $k = 1{:}N$
3. If k mod $2 == 0$, then the chromosome k is a male, else the chromosome k is a female
4. Select females to be crossed by tournament selection with tournament size t
5. For each selected female
6. Select randomly t males
7. For $j = 1{:}t$ males
8. Calculate the MD_j = Manhattan distance (X_{female} , X_j) via $MD_j = \sum_{i=1,\dots,n} |x_{female,i} - x_{ji}|$
9. Select the male having a maximum Manhattan distance $MD_l = \max_j MD_j$
10. If $card\{MD_l\} \geq 2$, then calculate the fitness value $f(X_j) = \sum_{i=1}^{n} p_i x_{ji}$
11. Select the male with the highest fitness value $f_l = \max_j \{f(X_j)\}$
12. If $card\{f_l\} \geq 2$, then calculate the weight of items for each male chromosome $W_j = \sum_{i=1}^{n} \sum_{d=1}^{m} w_{di}$
13. Select the male with the lightest items $W_l = \min_j W_j$
14. If $card\{W_l\} \geq 2$, then choose the male chromosome randomly

---

[19] The disassortative mating is a mating pattern that consists of choosing dissimilar individuals in genotype or phenotype space, in order to reduce the similarities within family.

15. End for
16. End for
17. End for
18. **Output: Pairs of chromosomes with opposite sex**

Algorithm 3. The pseudo code of the proposed sexual selection strategy

Other variants of the proposed sexual selection may be effective by combining some of the proposed preference criteria with the ones proposed by (Varnamkhasti & Lee, 2012a). In the first variant, the female chromosome chooses its partner according to Manhattan distance, or the fitness value, or active genes. However, in the second variant, the mate preference is based on Hamming distance, or fitness value, or the total weight corresponding to candidate male chromosomes. Henceforth, Varnamakhasti and Lee's sexual selection strategy is denoted as SS and our proposed strategy is signed by ISS (as an abbreviation for Improved Sexual Selection), whereas the two aforementioned variants of ISS are baptized respectively ISS* and ISS**.

3.4 The proposed two-stage recombination operator

As the above-mentioned sexual selection strategies rest on disassortative mating, we assume that it will be promising to apply an adequate crossover operator that takes profit from dissimilarities between parents. Such a crossover operator is the one suggested by (Aghezzaf & Naimi, 2009). This operator called "two-stage recombination operator" seems to be convenient to realize an effective outcrossing.

Let us consider two parents with 5 candidate items to be packed into the knapsack with 2 dimensions with capacities $c_1 = 20$ and $c_2 = 30$. The instance of the problem is represented by the triplet (w_{i1}, w_{i2}, p_i): (5,7,15); (4,11,12); (3,3,8); (5,8,20); (10,16,13)

	Step 1: {1 0 1 ? ?}
Parent 1: {1 0 1 1 0} Parent 2: {1 0 1 0 1}	**Step 2:** We sort the items 4 and 5 in decreasing order of their fitness values: $\frac{20}{8+5} > \frac{13}{10+16}$ Then we iteratively insert the items by taking into account the capacity constraints Offspring: {1 0 1 1 0}

Figure 9. An illustrative example of the two-stage recombination operator

The basic framework of the two-stage recombination operator consists of producing only one offspring through two fairly phases (see Figure 9 above). In the first phase or "the genetic shared-information" phase, the offspring inherits the parents' similar genes (*i.e.* the genes having the same alleles at the same locus in both parents) in order to preserve their common genetic information. Then, in the second phase or "the problem fitness-information" phase, the parents' non-similar genes (*i.e.* the genes having different alleles at the same locus) are selected

from one of both parents according to a gene preference criterion. In fact, during the second phase, the authors attribute to non-similar genes (also called the non-common items in the context of knapsack problem) a fitness function that favors the corresponding items with higher profit and lower weight. Knowing that this specific knowledge crossover operator is firstly developed for solving the multi-objective version of 0/1 MKP, it is however possible to adapt it in the context of 0/1 MKP by making a slight modification. A potential change consists of using a gene's fitness function that depends on the structure of 0/1 MKP, we have herein used the fitness function represented by equation (3.5). Such adaptation gives rise to a first variant denoted by 2SR_v1. The pseudo-code of the 2SR_v1 operator is given in Algorithm 4.

1. **Input: A couple of chromosome mother $X_{mother} = \{x_{mother,1}, x_{mother,2}, \ldots, x_{mother,n}\}$ and chromosome father $X_{father} = \{x_{father,1}, x_{father,2}, \ldots, x_{father,n}\}$**
2. $O = X_{father} \otimes X_{mother}$ (where $\otimes$ is a symbol of crossover and $O = \{o_1, o_2, \ldots, o_n\}$ is the generated offspring)
3. ----------Phase 1----------
4. For $i = 1{:}\,n$
5. If $x_{mother,i} == x_{father,i}$ then $o_i = x_{father,i}$
6. Else create a list of non-common items
7. End for
8. Calculate $R_j = \sum_{i=1}^{n} w_{ij}\, x_i$ (R_j is the accumulated resources of constraint j)
9. ----------Phase 2----------
10. For each l in the list of non-common items
11. Calculate $f_l = \frac{p_l}{\sum_{j=1}^{m} w_{ij}}$
12. Sort f_l in decreasing order as well as its corresponding item l
13. While $R_j \leq c_j$, $\forall j = 1, \ldots, m$
14. If $R_j + w_{lj} \leq c_j$ then $o_l = 1$
15. $R_j = R_j + w_{lj}$
16. Else $o_l = 0$
17. $l = l + 1$
18. End For
19. **Output : feasible offspring chromosomes**

Algorithm 4. The pseudo code of the 2SR_v1 operator

As mentioned before, the two-stage recombination operator and its variant depend on a fitness-based criterion to build new chromosomes. Consequently, the search space will be biased towards fitter solutions, thereby pushing the search process towards the intensification strategy. So, to keep genetic diversity after outbreeding, we propose another variant of the two-stage recombination operator that injects a degree of randomness, especially in the second phase, thereby ensuring that items with higher fitness will not always be chosen which gives the chance to items with lower fitness to participate in building solutions. This second variant of the two-stage recombination operator is denoted by 2SR_v2. Its pseudo-code is presented in the following algorithm (Algorithm 5).

1. **Input: A couple of chromosome mother $X_{mother} = \{x_{mother,1}, x_{mother,2}, \ldots, x_{mother,n}\}$ and chromosome father $X_{father} = \{x_{father,1}, x_{father,2}, \ldots, x_{father,n}\}$**
2. $O = X_{father} \otimes X_{mother}$
3. ----------Phase 1----------
4. For $i = 1{:}\,n$
5. If $x_{mother,i} == x_{father,i}$ then $o_i = x_{mother,i}$
6. else create a list of non-common items
7. End for
8. Calculate $R_j = \sum_{i=1}^{n} w_{ij}\, x_i$ (R_j is the accumulated resources of constraint j)
9. ----------Phase 2----------
10. For each l in the list of non-common items
11. While $R_j \leq c_j, \forall j = 1, \ldots, m$ do
12. Generate a random number $rand \in [0, 1]$
13. If $rand > 0{,}5$ and $R_j + w_{lj} \leq c_j$, then $o_l = 1$
14. $R_j = R_j + w_{lj}$
15. Else $o_l = 0$
16. $l = l + 1$
17. End For
18. **Output: Feasible offspring chromosomes**

Algorithm 5. The pseudo code of the 2SR_v2 operator

Once the crossover operation is done, a mutation operation is applied in some offspring chromosomes with a probability p_m in order to change their genetic composition. In the proposed algorithm, we apply the bit-flip mutation defined above (see section 2). Then, the resulting offspring chromosomes are ranked in a new population by using the generational replacement strategy. The algorithm is stopped when a preset termination criterion is met.

3.5 Flowchart of the proposed GA

Henceforth, the genetic algorithm resulting from a marriage of the aforementioned proposed operators is called "improved sexual genetic algorithm (ISGA)". A flowchart of the ISGA is outlined in Figure 10 in order to summarize their main steps.

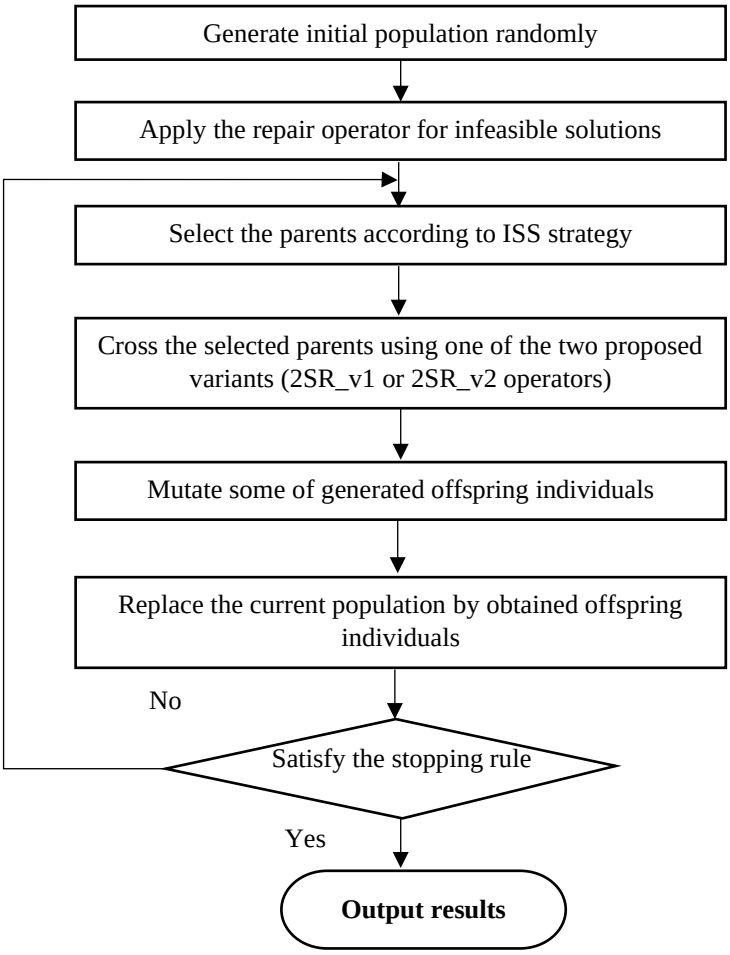


Figure 10. The flowchart of ISGA

4 Experimental results

4.1 Simulation with a fixed number of generations

In order to test the performance of the two proposed genetic operators (the sexual selection and the two-stage recombination operators), two popular datasets of 0/1 MKP instances are used. The first dataset is the ORLib dataset, proposed by (Chu & Beasley, 1998) and available at the URL[20] below. It contains m ($\in$ {5, 10, 30}) dimensions and n ($\in$ {100, 250, 500}) items. In this dataset, 30 test problems were generated for each $m - n$ combination; where the first ten problems correspond to a tightness ratio α = 0.25, the next ten problems correspond to α = 0.50 and the latter ten problems correspond to α = 0.75. The second dataset is the GK dataset, proposed by Glover and Kochenberger in 2000 (see (Drake, et al., 2016)) where the number of dimensions m is between 15 and 100, while the number of items n ranging between 100 and 2500. The ISGA as well as the comparative algorithms are implemented in JAVA programming language using an Intel® Core™ i5 computer with 2.5 GHz and 4.0 Go of RAM, running on 64-bits Windows 7 operating system. Nevertheless, before evaluating the

[20] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html

performance of ISGA, we will start by comparing each of the proposed operators against other operators commonly used in the literature. The parameters used in all experiments are given in Table 2. We should note that the values of these parameters are set through a preliminary phase of intensive empirical tests.

Table 2. Parameters setting

Parameters	Values
Population size	50
Crossover probability	0,7
Mutation probability	1/(Number of items)
Maximum generation	1000

This results section is divided into two parts. First, we present and discuss the results related to the proposed sexual selection operator. Then, we present and discuss the results related to each of the two proposed variants of the two-stage recombination operator.

4.1.1 Results of the proposed sexual selection

As pointed out above, sexual selection enhances the search ability while balancing between both intensification and diversification. To clarify the influence of the sexual selection strategy on the performance of GA, it is tested on 27 problem instances from ORLib dataset stated as small to large-scaled problems and four problem instances belonging to GK dataset that represent more difficult and large-scaled problems. For the sake of clarity, the ORLib instances are named ORmxn-α-i, where m is the number of constraints, n is the number of items, α is the tightness ratio and i is the index of instances. Only one instance is randomly chosen to represent ten problem instances having the same tightness ratio. Then the performance of the new selection strategy (ISS) is compared against that of the sexual selection mechanism (SS) proposed by (Varnamkhasti & Lee, 2012a) as well as four other classical selection operators; namely Binary Tournament Selection (TS), Roulette Wheel Selection (RWS), Statistical Universal Selection (SUS) and Truncation Selection (TrS). For this simulation, the uniform crossover (UC) is chosen as a default crossover operator to breed chromosome parents. This choice was motivated by the fact that UC is expected to perform better than other standard crossover operators, especially the one-point crossover and the two-point crossover.

The comparison results are depicted in Tables 3-6. The first column in Tables 3-5 indicates representative instances. The second column reports the best found solutions for ORLib dataset. Computational results of the GA using the ISS, on one hand, and the GA using each of the five comparative selection methods, on the other hand, are shown in the remaining columns of Tables 3-5. In Table 6, the two first columns present respectively the test instances and their sizes. The third column reports the best solutions achieved for GK dataset, while the remaining columns depict the outcome results of ISS strategy and those achieved by the comparative selection operators. For each test instance, the comparison is performed based on two

performance measures considering the mean results of 30 independent runs. In fact, we used the average execution time (in seconds) and the percentage gap that is calculated as follows:

$$\text{Gap\%} = \frac{optimal\ solution - achieved\ solution}{optimal\ solution} \times 100 \qquad (3.7)$$

Where "*achieved solution*" is the average fitness value, and "*optimal solution*" is the best known solution for the 0/1 MKP instances (available at ORLib site); it is the upper bound provided by resolving the LP relaxation of 0/1 MKP. In tables 3-6, the values in bold characters indicate the best results among all considered algorithms. From these tables, we can observe that ISS obtains best results in terms of solutions quality for the majority of tackled instances except for three instances OR30x100-0.75-27, OR30x250-0.75-28, and OR30x500-0.75-27 in which the SUS operator is better than the other considered selection operators including ISS. The comparison results have the same tendency in terms of solution quality for small to large-scaled instances. As far as computing time is concerned, the results are obtained in reasonable execution time. However, for large instances, we remark a slightly significant difference between the execution time of the proposed selection method and that consumed by the comparative selection methods in favor of the latter.

Table 3. Simulation results of the proposed sexual selection using Chu and Beasely instances (*n*=100)

Instances	Best Known		ISS	SS	TS	RWS	TrS	SUS
OR5x100-0.25-8	23410	Gap%	**2,65**	2,71	2,99	3,01	3,39	3,21
		CPU	0,17	0,12	0,10	0,10	0,10	0,10
OR5x100-0.50-13	41968	Gap%	**2,24**	2,58	2,68	2,68	2,90	2,50
		CPU	0,17	0,13	0,10	0,10	0,10	0,10
OR5x100-0.75-21	59822	Gap%	**1,55**	1,62	**1,55**	2,03	1,74	1,56
		CPU	0,16	0,11	0,10	0,10	0,09	0,10
OR10x100-0.25-2	22801	Gap%	**4,02**	4,22	4,57	4,50	5,02	4,27
		CPU	0,25	0,17	0,15	0,14	0,15	0,15
OR10x100-0.50-12	42344	Gap%	**3,08**	3,42	3,38	3,50	3,81	3,29
		CPU	0,23	0,15	0,14	0,14	0,15	0,15
OR10x100-0.75-28	59391	Gap%	**2,28**	2,50	2,86	2,84	3,05	2,84
		CPU	0,23	0,15	0,14	0,14	0,14	0,15
OR30x100-0.25-5	21844	Gap%	**4,59**	4,82	5,33	4,98	5,20	5,20
		CPU	0,53	0,33	0,33	0,36	0,34	0,34
OR30x100-0.50-19	42230	Gap%	**3,34**	**3,34**	3,58	3,68	4,33	3,52
		CPU	0,50	0,30	0,38	0,34	0,30	0,37
OR30x100-0.75-27	58132	Gap%	2,05	2,40	2,26	2,48	2,93	**1,92**
		CPU	0,48	0,30	0,32	0,34	0,30	0,32

Table 4. Simulation results of the proposed sexual selection using Chu and Beasely instances (n =250)

Instances	Best Known		ISS	SS	TS	RWS	TrS	SUS
OR5x250-0.25-9	61885	Gap%	**2,41**	2,48	2,98	2,67	3,40	2,84
		CPU	0,47	0,31	0,31	0,32	0,32	0,31
OR5x250-0.50-17	109040	Gap%	**2,28**	2,44	2,70	2,87	3,01	2,63
		CPU	0,45	0,30	0,29	0,30	0,28	0,31
OR5x250-0.75-21	149665	Gap%	**2,47**	2,96	2,94	3,61	3,55	2,75
		CPU	0,16	0,11	0,10	0,10	0,09	0,10
OR10x250-0.25-8	58933	Gap%	**3,48**	3,89	4,05	3,70	4,78	4,12
		CPU	0,64	0,45	0,44	0,45	0,43	0,45
OR10x250-0.50-20	106723	Gap%	**3,02**	3,08	3,30	3,55	4,02	3,09
		CPU	0,66	0,48	0,43	0,41	0,45	0,42
OR10x250-0.75-25	151966	Gap%	**3,10**	3,28	3,52	4,15	3,94	3,20
		CPU	0,63	0,55	0,44	0,42	0,37	0,42
OR30x250-0.25-10	56447	Gap%	**4,70**	4,78	5,28	5,15	5,65	4,92
		CPU	1,33	0,95	1,14	1,15	1,11	1,00
OR30x250-0.50-15	107414	Gap%	**3,17**	3,41	3,60	4,04	4,66	3,29
		CPU	1,33	0,94	0,98	1,20	0,96	0,99
OR30x250-0.75-28	152912	Gap%	3,06	3,17	3,54	3,94	4,15	**2,94**
		CPU	1,31	0,92	0,92	1,01	0,92	0,96

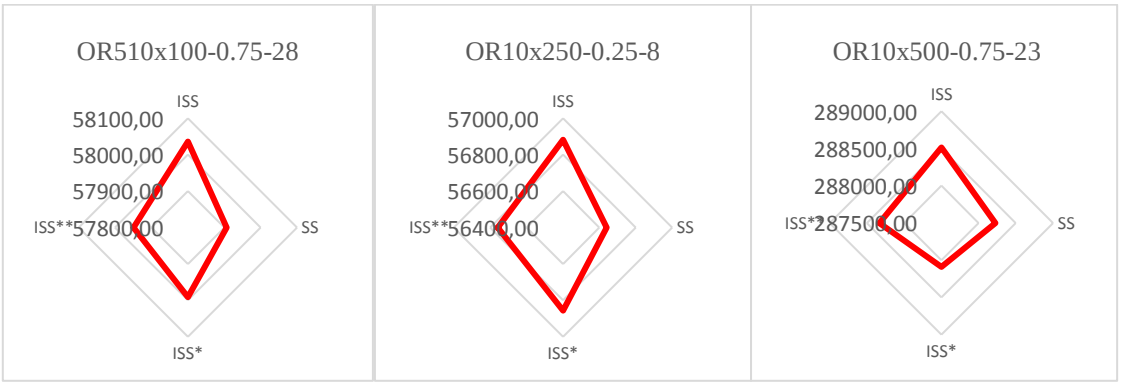
Table 5. Simulation results of the proposed sexual selection using Chu and Beasely instances (n =500)

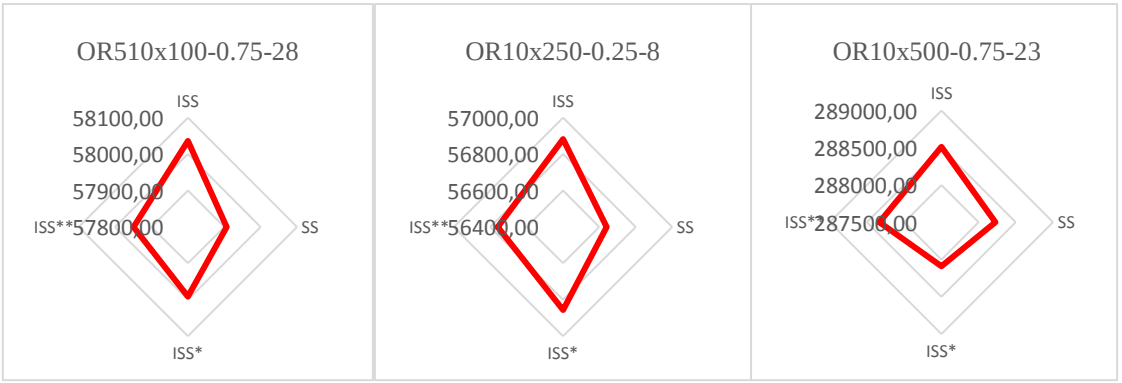
Instances	Best Known		ISS	SS	TS	RWS	TrS	SUS
OR5x500-0.25-2	117879	Gap%	**2,94**	2,99	3,49	3,41	4,15	3,18
		CPU	1,14	0,76	0,75	0,75	0,67	0,68
OR5x500-0.50-11	218428	Gap%	**3,34**	3,35	3,69	3,89	4,45	3,42
		CPU	1,06	0,60	0,65	0,64	0,56	0,62
OR5x500-0.75-22	308086	Gap%	**4,61**	5,12	4,90	5,73	5,65	4,78
		CPU	0,97	0,58	0,55	0,55	0,51	0,53
OR10x500-0.25-7	118329	Gap%	**2,27**	2,33	2,74	3,03	2,83	2,47
		CPU	1,34	1,02	0,94	1,00	1,02	1,13
OR10x500-0.50-14	213859	Gap%	**2,08**	2,31	2,75	3,21	3,41	2,57
		CPU	1,21	0,93	0,81	0,90	0,90	0,95
OR10x500-0.75-23	300757	Gap%	**4,07**	4,17	4,42	5,10	4,66	4,15
		CPU	1,19	0,75	0,77	0,80	0,85	0,83
OR30x500-0.25-6	115741	Gap%	**3,80**	4,11	4,43	4,45	5,24	3,97
		CPU	2,92	1,95	2,62	2,10	2,10	2,18
OR30x500-0.50-11	218104	Gap%	**3,61**	3,86	4,00	4,35	4,81	3,76
		CPU	2,95	1,91	2,40	1,95	1,93	2,03
OR30x500-0.75-27	303364	Gap%	4,74	5,13	5,13	6,00	5,31	**4,65**
		CPU	2,74	1,77	1,92	1,86	1,83	1,92

Table 6. Simulation results of the proposed sexual selection using the large instances of Glover and Kochenberger

Instances	m x n	Best Known		ISS	SS	TS	RWS	TrS	SUS
gk08	50 x 500	18801	Gap%	**1,76**	1,90	2,10	1,93	2,14	1,92
			CPU	3,96	2,81	2,88	2,90	3,04	2,98
gk09	25 x 1500	58085	Gap%	**1,98**	2,08	2,20	2,24	2,52	2,16
			CPU	7,79	5,24	5,57	5,70	5,19	5,48
gk10	50 x 1500	57292	Gap%	**1,73**	1,78	1,98	1,91	1,99	1,77
			CPU	14,94	11,06	11,79	11,72	11,43	11,94
gk11	100 x 2500	95231	Gap%	**1,58**	1,63	1,74	1,65	1,79	1,63
			CPU	48,75	40,16	41,62	40,38	37,16	41,49

Figures 11 and 12 represent a comparison of the average fitness of ISS, SS, and the two variants of ISS (defined in subsection 3.3) using six test instances from ORLib dataset that have different characteristics, in addition to some high-dimensional instances from GK dataset. The idea behind this comparison is to highlight the togetherness between the mating criteria used in ISS against those of SS, ISS*, and ISS**. It can be shown from radar charts that the ISS axis has a point far away from the center in comparison with the three other variants of sexual selection, which indicates that the ISS strategy is more effective in terms of the quality of solutions. Moreover, it can be clearly seen that its effectiveness is worthwhile for all considered test instances independently of their sizes or their tightness ratios.





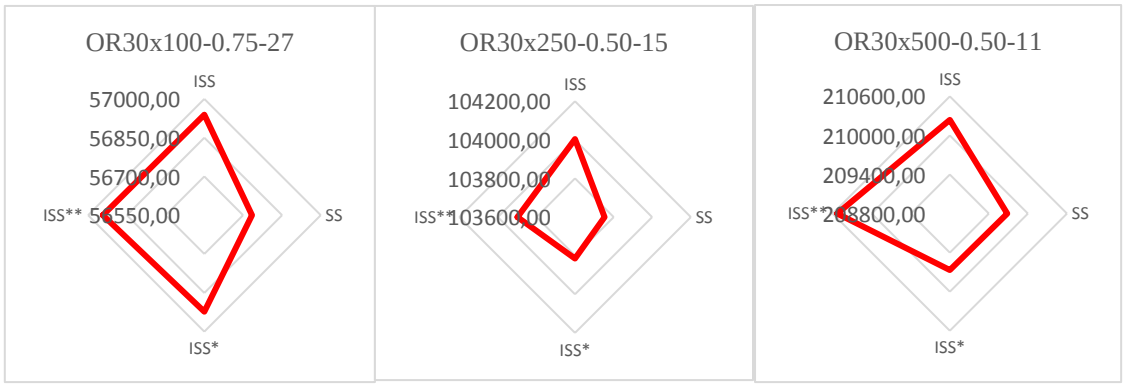


Figure 11. Comparative results of sexual selection strategies using Chu and Beasely instances

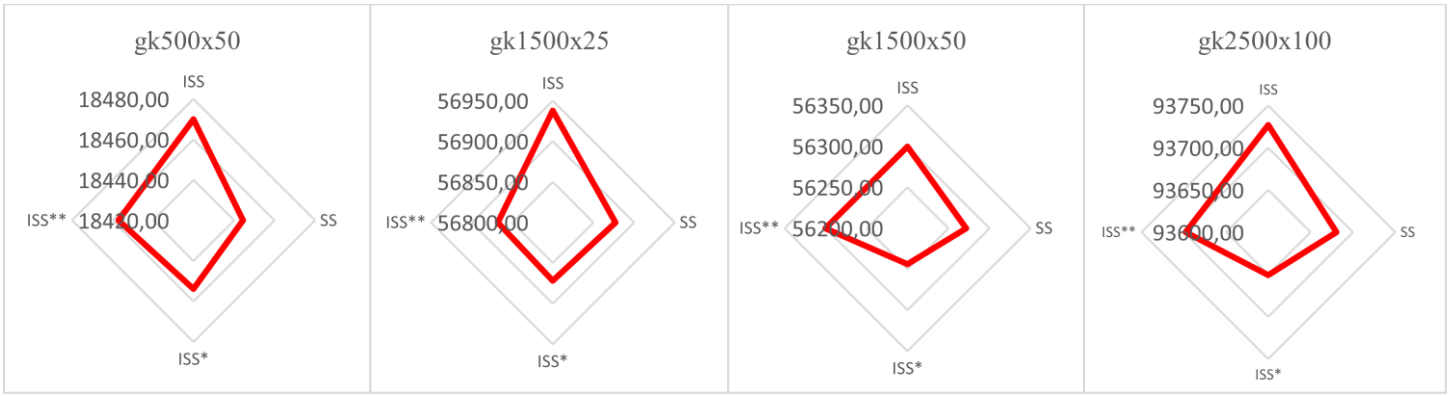


Figure 12. Comparative results of sexual selection strategies using Glover and Kochenberger instances

4.1.2 Results of the proposed two-stage recombination operator

To evaluate both proposed variants of the two-stage recombination operator (2SR_v1 and 2SR_v2), we used 27 test problems of 0/1 MKP from the ORLib dataset. These test problems were randomly selected so that they are completely different from the ones used in the previous subsection, in order to avoid redundancy of obtained results when we combine uniform crossover with the selection approaches. The outcome results are compared to three standard crossover operators, namely the one-point crossover operator (1PC), two-point crossover operator (2PC), and uniform crossover operator (UC). The results are shown in Tables 7-9 in terms of the percentage gap and the runtime for each of the five crossover operators combined with three selection strategies that are ISS, SS, and TS giving thus a total of 15 GAs. These comparison experiments have three main objectives: The first is to prove the robustness of 2SR_v1 and 2SR_v2 against the traditional crossover operators. The second objective is to compare the behavior of 2SR_v1 and its rival 2SR_v2, while the last objective consists of investigating the effectiveness and robustness of the ISGA combining the ISS strategy with each of both operators 2SR_v1 and 2SR_v2.

Table 7. Comparative results of the two-stage recombination operators on small MKP instances (n = 100)

Instances	Best Known	Crossover operator	TS		SS		ISS	
			Gap%	CPU	Gap%	CPU	Gap%	CPU
OR5x100-0.25-4	23534	1PC	6,31	0,07	6,03	0,09	6,07	0,18
		2PC	5,99	0,08	5,42	0,10	5,72	0,14
		UC	4,37	0,10	4,20	0,12	4,68	0,19
		2SR_v1	4,81	0,07	5,37	0,11	5,12	0,16
		2SR_v2	**1,36**	0,06	**1,11**	0,11	**1,06**	0,16
OR5x100-0.50-14	45090	1PC	5,73	0,07	5,43	0,08	5,45	0,17
		2PC	4,99	0,07	4,61	0,08	4,81	0,13
		UC	2,09	0,10	1,98	0,12	1,70	0,16
		2SR_v1	2,21	0,04	2,35	0,06	2,46	0,12
		2SR_v2	**1,93**	0,07	**1,65**	0,10	**1,61**	0,15
OR5x100-0.75-22	62081	1PC	7,55	0,06	6,97	0,08	6,89	0,16
		2PC	6,44	0,06	5,91	0,08	5,85	0,13
		UC	2,01	0,10	2,24	0,11	2,03	0,16
		2SR_v1	**0,79**	0,04	**0,75**	0,05	**0,78**	0,11
		2SR_v2	1,21	0,06	1,06	0,08	1,09	0,13
OR10x100-0.25-8	22635	1PC	6,76	0,11	5,89	0,13	6,52	0,19
		2PC	6,27	0,12	6,33	0,12	5,66	0,22
		UC	4,49	0,15	4,21	0,15	3,92	0,23
		2SR_v1	5,07	0,05	5,26	0,11	5,29	0,19
		2SR_v2	**1,83**	0,06	**1,66**	0,11	**1,54**	0,19
OR10x100-0.50-18	42970	1PC	6,30	0,11	5,90	0,12	5,66	0,20
		2PC	5,90	0,11	5,48	0,12	4,91	0,20
		UC	2,66	0,14	2,27	0,16	2,48	0,23
		2SR_v1	2,20	0,04	2,14	0,07	2,12	0,15
		2SR_v2	**2,11**	0,08	**1,67**	0,12	**1,66**	0,19
OR10x100-0.75-22	58978	1PC	7,44	0,10	7,39	0,11	7,07	0,21
		2PC	6,87	0,10	6,04	0,11	6,16	0,21
		UC	2,97	0,14	2,62	0,16	2,43	0,23
		2SR_v1	1,48	0,05	1,54	0,07	1,49	0,15
		2SR_v2	**1,41**	0,09	**1,16**	0,11	**1,19**	0,19
OR30x100-0.25-2	21716	1PC	7,50	0,30	6,62	0,30	6,70	0,47
		2PC	6,78	0,31	6,95	0,31	6,75	0,47
		UC	5,22	0,34	4,65	0,37	4,68	0,52
		2SR_v1	6,14	0,10	6,57	0,25	6,36	0,41
		2SR_v2	**2,56**	0,11	**1,92**	0,25	**2,15**	0,44
OR30x100-0.50-16	41058	1PC	7,63	0,28	6,58	0,30	7,36	0,45
		2PC	6,88	0,27	6,81	0,28	6,45	0,47
		UC	4,37	0,31	4,20	0,32	4,14	0,49
		2SR_v1	4,05	0,11	4,15	0,21	3,93	0,35
		2SR_v2	**2,35**	0,18	**1,89**	0,28	**1,69**	0,43
OR30x100-0.75-30	60603	1PC	7,09	0,26	6,97	0,26	6,87	0,45
		2PC	6,15	0,27	6,00	0,29	5,90	0,42

		UC	2,62	0,32	2,50	0,31	2,79	0,46
		2SR_v1	**1,66**	0,09	1,67	0,14	1,70	0,30
		2SR_v2	1,83	0,18	**1,56**	0,24	**1,56**	0,39

Table 8. Comparative results of the two-stage recombination operators on medium MKP instances (n = 250)

Instances	Best Known	Crossover operator	TS		SS		ISS	
			Gap%	CPU	Gap%	CPU	Gap%	CPU
OR5x250-0.25-4	59463	1PC	5,61	0,22	5,33	0,56	5,37	0,41
		2PC	5,16	0,23	5,35	0,24	4,84	0,41
		UC	3,05	0,32	2,83	0,32	2,95	0,48
		2SR_v1	3,18	0,22	3,32	0,38	3,21	0,52
		2SR_v2	**1,48**	0,37	**1,17**	0,61	**1,22**	0,77
OR5x250-0.50-18	109042	1PC	7,39	0,19	6,86	0,34	7,51	0,37
		2PC	6,71	0,19	5,96	0,21	6,22	0,36
		UC	2,56	0,30	2,50	0,30	2,35	0,45
		2SR_v1	**1,95**	0,13	**1,91**	0,17	**1,94**	0,33
		2SR_v2	2,53	0,50	2,35	0,66	2,13	0,82
OR5x250-0.75-23	149334	1PC	14,20	0,17	13,63	0,18	12,49	0,35
		2PC	11,89	0,18	11,47	0,18	10,60	0,35
		UC	2,95	0,28	2,99	0,29	2,78	0,44
		2SR_v1	**0,60**	0,16	**0,64**	0,18	**0,63**	0,35
		2SR_v2	1,74	0,46	1,56	0,53	1,52	0,67
OR10x250-0.25-6	58824	1PC	7,49	0,36	6,51	0,37	6,93	0,63
		2PC	7,18	0,38	6,80	0,38	6,49	0,65
		UC	4,54	0,44	4,09	0,47	4,32	0,70
		2SR_v1	5,66	0,40	5,49	0,79	5,65	0,97
		2SR_v2	**1,84**	0,43	**1,56**	0,77	**1,52**	1,02
OR10x250-0.50-14	110086	1PC	8,33	0,33	7,52	0,36	7,13	0,64
		2PC	7,09	0,35	6,63	0,37	6,76	0,63
		UC	3,31	0,43	3,05	0,44	3,23	0,67
		2SR_v1	3,14	0,21	3,34	0,29	3,34	0,51
		2SR_v2	**2,97**	0,76	**2,49**	1,03	**2,46**	1,24
OR10x250-0.75-23	151909	1PC	13,83	0,32	13,52	0,37	12,90	0,52
		2PC	12,48	0,32	10,70	0,33	11,35	0,54
		UC	3,63	0,44	3,14	0,45	3,03	0,66
		2SR_v1	**0,93**	0,21	**0,92**	0,23	**0,90**	0,45
		2SR_v2	2,04	0,71	1,79	0,82	1,75	0,91
OR30x250-0.25-8	56457	1PC	8,21	0,88	7,82	0,84	7,56	1,36
		2PC	8,08	0,89	7,27	0,99	7,11	1,25
		UC	5,37	1,04	4,75	0,96	4,86	1,36
		2SR_v1	5,78	0,39	5,72	1,16	5,82	1,50
		2SR_v2	**2,36**	1,02	**2,07**	1,95	**1,99**	2,31
OR30x250-0.50-20	105780	1PC	8,33	0,86	8,06	0,82	7,78	1,56
		2PC	7,69	0,82	7,26	0,92	7,29	1,20
		UC	3,57	1,01	3,14	0,94	3,36	1,32

		2SR_v1	3,69	0,38	3,62	0,54	3,65	0,93
		2SR_v2	**3,41**	1,70	**2,94**	2,40	**2,91**	2,61
OR30x250-0.75-30	149668	1PC	14,65	0,83	13,18	0,80	13,07	1,29
		2PC	12,71	0,79	11,43	0,78	11,15	1,34
		UC	4,04	0,95	3,65	0,92	3,81	1,34
		2SR_v1	**1,69**	0,58	**1,71**	0,73	**1,79**	1,08
		2SR_v2	2,29	1,64	1,96	2,11	2,06	2,57

Table 9. Comparative results of the two-stage recombination operators on large MKP instances (n = 500)

Instances	Best Known	Crossover	TS		SS		ISS	
			Gap%	CPU	Gap%	CPU	Gap%	CPU
OR5x500-0.25-9	121575	1PC	6,86	0,57	6,43	0,50	6,28	0,85
		2PC	6,42	0,50	6,20	0,51	6,08	0,90
		UC	4,02	0,76	3,93	0,68	4,03	1,12
		2SR_v1	3,63	0,51	3,56	1,14	3,65	1,51
		2SR_v2	**1,94**	1,59	**1,70**	2,44	**1,53**	2,77
OR5x500-0.50-12	221202	1PC	8,82	0,41	9,10	0,39	8,54	0,76
		2PC	8,60	0,40	8,25	0,41	8,08	0,73
		UC	3,70	0,68	3,38	0,61	3,37	1,04
		2SR_v1	**2,01**	0,35	**2,00**	0,45	**2,00**	0,82
		2SR_v2	4,39	2,24	3,72	3,02	3,89	3,41
OR5x500-0.75-25	300342	1PC	21,53	0,34	21,34	0,35	21,06	0,66
		2PC	20,16	0,36	18,84	0,35	18,29	0,74
		UC	4,53	0,56	4,71	0,60	4,72	0,98
		2SR_v1	**0,47**	0,49	**0,46**	0,62	**0,44**	0,91
		2SR_v2	2,37	1,73	2,09	2,24	2,15	2,50
OR10x500-0.25-5	116509	1PC	6,45	0,80	6,04	0,88	6,36	1,17
		2PC	6,32	0,76	6,09	0,90	6,00	1,37
		UC	4,17	0,98	3,87	1,04	3,67	1,36
		2SR_v1	3,28	0,74	3,33	1,87	3,29	2,30
		2SR_v2	**2,35**	2,35	**2,06**	3,92	**1,86**	4,26
OR10x500-0.50-15	213859	1PC	9,15	0,68	8,83	0,77	8,99	1,01
		2PC	8,83	0,62	8,72	0,73	8,23	1,22
		UC	4,08	0,84	3,71	0,98	3,68	1,33
		2SR_v1	**1,92**	0,60	**1,81**	0,99	**1,85**	1,26
		2SR_v2	4,50	3,58	3,72	4,82	3,89	5,17
OR10x500-0.75-26	301836	1PC	21,21	0,58	19,53	0,57	21,19	0,94
		2PC	18,61	0,55	17,59	0,61	18,35	0,95
		UC	4,65	0,75	4,57	0,74	4,83	1,13
		2SR_v1	**0,97**	0,57	**0,96**	0,87	**0,96**	1,16
		2SR_v2	2,67	2,79	2,32	3,85	2,45	3,78
OR30x500-0.25-10	117116	1PC	7,06	1,93	6,70	1,74	6,62	2,60
		2PC	6,79	1,96	6,34	1,84	6,62	2,60
		UC	4,76	2,40	4,36	1,96	4,12	3,00
		2SR_v1	4,08	1,60	4,15	4,57	4,19	5,09

		2SR_v2	**2,72**	4,64	**2,28**	8,83	**2,32**	9,39
OR30x500-0.50-14	217910	1PC	9,50	1,80	9,36	1,63	9,38	2,49
		2PC	8,90	1,77	8,85	1,74	8,73	2,55
		UC	4,59	2,04	4,14	1,92	4,25	3,34
		2SR_v1	**2,73**	1,06	**2,72**	1,80	**2,65**	2,86
		2SR_v2	5,06	8,18	4,17	12,20	4,30	11,78
OR30x500-0.75-25	304462	1PC	20,64	1,64	20,35	1,53	20,62	2,37
		2PC	19,42	1,70	18,42	1,58	18,59	2,39
		UC	5,07	1,88	5,22	1,79	5,00	2,63
		2SR_v1	**1,01**	1,33	**1,03**	1,70	**1,01**	2,81
		2SR_v2	3,05	7,03	2,70	9,00	2,71	9,91

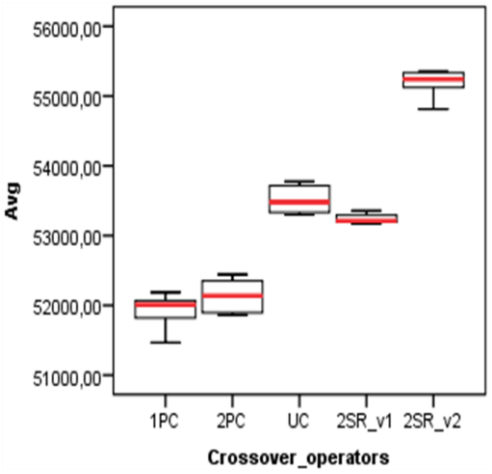
From Table 7, we can see that for small-scaled instances (n = 100), the 2SR_v2 operator outperforms the 2SR_v1 operator as well as the three classical crossover operators in closely similar computing time, except in the instances having a tightness ratio of 0.75. Table 8 shows that for medium scaled-instances (n = 250), the 2SR_v2 operator remains better than 2SR_v1 and the classical crossover operators in terms of solution quality, with an exception for the instances having a tightness ratio of 0.75. However, the 2SR_v2 operator consumes a lot of time in comparison with that consumed by the other comparative crossover operators including 2SR_v1. Table 9 shows the results corresponding to large scaled-instances (n = 500). From this table, it can be observed that the performance of 2SR_v2 downgrades in comparison with 2SR_v1 in both terms of solution quality and execution time, mainly when the tightness ratio equals 0.50 or 0.75. Nevertheless, 2SR_v2 performs well in terms of solution quality for the instances having a tightness ratio of 0.25 when compared to 2SR_v1. Moreover, it still achieves better solutions than the remaining comparative crossover operators regardless of the tightness ratios. However, it requires much more computing time than all other comparative crossover operators. Concerning the 2SR_v1 operator, it is outperformed by the uniform crossover operator in small and medium test instances, but it works better than this latter on large test instances. On the other hand, it can also be seen from Tables 7-9, that the 2SR_v1 operator resolves efficiently the 0/1 MKP when it is combined especially with ISS strategy, whilst the 2SR_v2 operator yields better results when it is combined with the SS strategy.

Furthermore, in order to show the benefit of incorporating randomness in a greedy construction heuristic, we report in Figure 13 the boxplots corresponding to the average fitness values obtained by the five comparative crossover operators for nine instances having a high knapsack dimension (= 30), different number of items, and different tightness ratios. From Figure 13, we can summarize what we have explained in the latter paragraph, namely the effectiveness of the 2SR_v2 operator depends on the tightness ratio, in particular when it is compared with the 2SR_v1 operator. Indeed, there is a significant difference between the fitness values of all crossover operators. For tightness ratios of 0.25 and 0.50, the corresponding median of the 2SR_v2 operator is higher than that of the 2SR_v1 operator and the other classical crossover operators. For a number of items exceeding 250 items, additionally to a tightness ratio of 0.75,

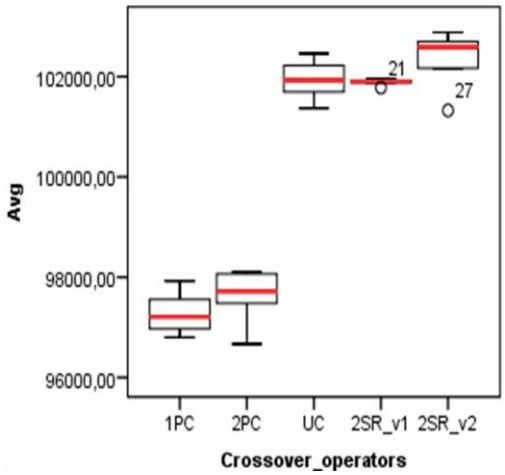
the corresponding median of the 2SR_v1 operator is greater than that of the 2SR_v2 operator, but the latter still has a greater median in comparison with classical crossover operators. This observation illustrates the fact that introducing a certain degree of randomness in the two-stage recombination operator can improve its efficiency, especially for small and medium instances with a tightness ratio of 0.25 or 0.50. It can also be seen that 2SR_v1 has a shorter height box in most instances, which means that it has a smaller variance. However, 2SR_v2 has different height boxes and hence a greater variance due to its random behavior.

To sum up, the 2SR_v2 operator performs successfully better than all comparative traditional crossover operators (1PC, 2PC, and UC) without any regard for size instances or tightness ratios, but it runs relatively slower than these latter operators; especially in large-scaled instances. Moreover, the 2SR_v1 operator is also better than the two traditional crossover operators (1PC and 2PC), except for UC that outperforms it in small and medium size instances. On the other hand, the comparison between 2SR_v1 and 2SR_v2 operators shows that the first outperforms the latter in large size instances with a higher tightness ratio (0.75), whereas the 2SR_v2 outperforms the 2SR_v1 in small and medium size instances with lower tightness ratios (0.25 or 0.50).

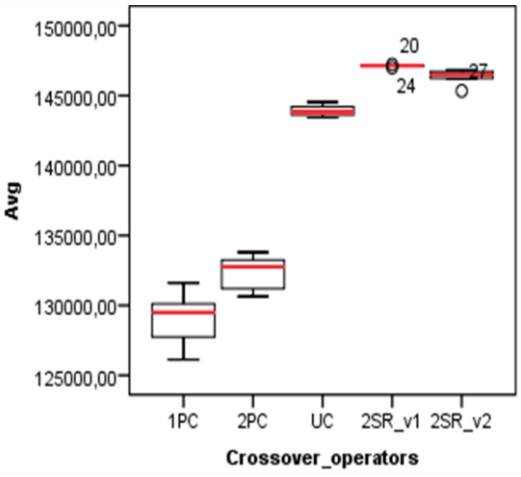
Nevertheless, we believe in the hypothesis that the results reached by 2SR_v1 and 2SR_v2 operators are still better than those reached by the other comparative operators if they are all executed in the same amount of time. We will demonstrate this assumption in the next subsection.



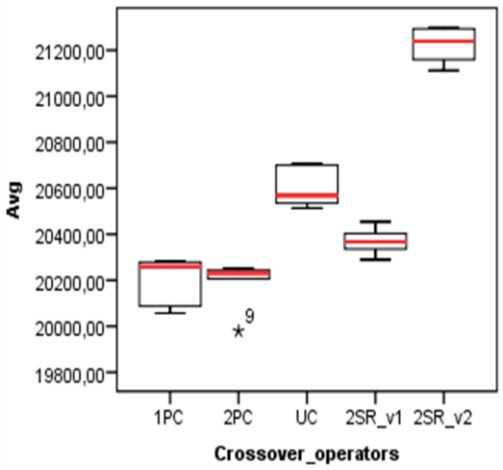
OR30x100-0.25-2



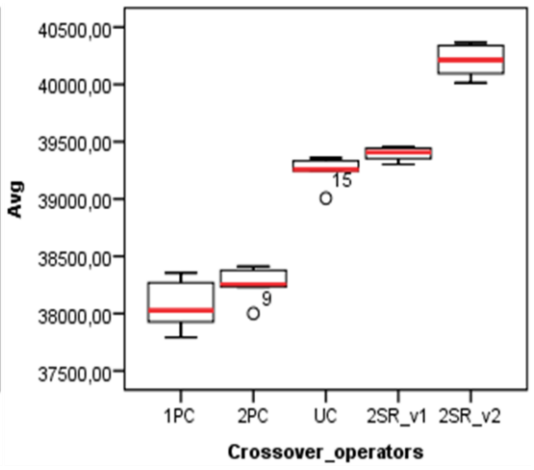
OR30x100-0.50-16



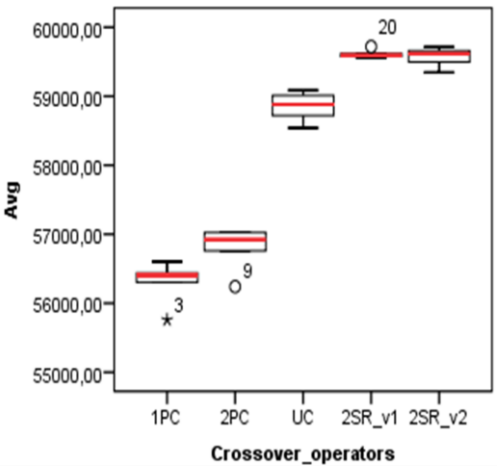
OR30x100-0.75-30



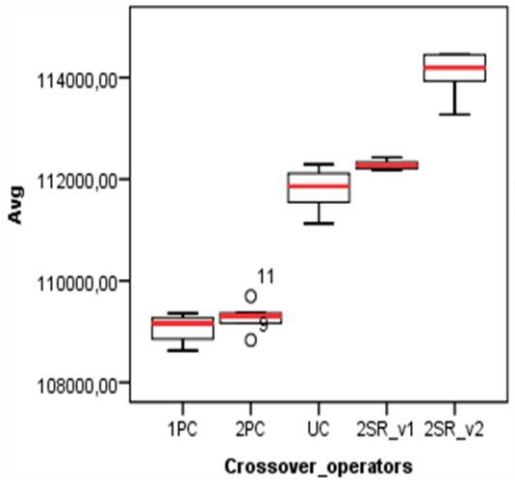
OR30x250-0.25-8



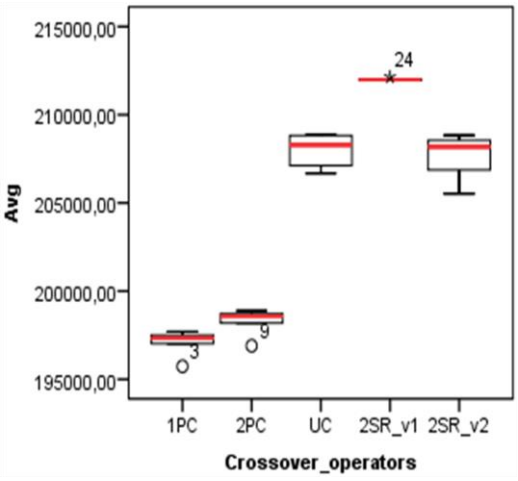
OR30x250-0.50-20



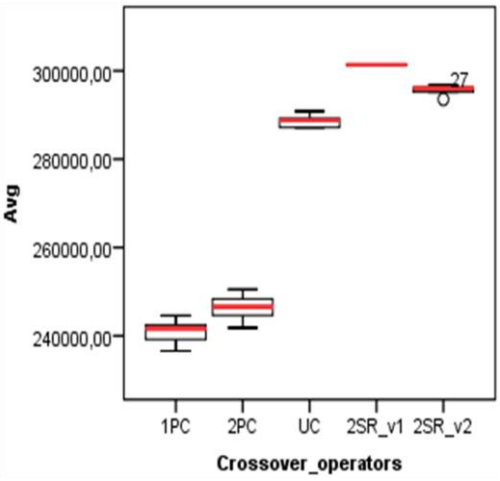
OR30x250-0.75-30



OR30x500-0.25-10



OR30x500-0.50-14



OR30x500-0.75-25

Figure 13. The average fitness value of five crossover operators upon nine representative Chu and Beasely instances

4.2 Simulation experiments with fixed computing time

4.2.1 Results of the proposed sexual selection

From the tables presented in the above sections, we can remark that the search speed is more notably affected by the number of items and not by the knapsack dimensions. Consequently, to test our proposed operators on 0/1 MKP benchmark instances, we have fixed an amount of time for each number of items regardless of the number of constraints (see Table 10 and 12).

Table 10. CPU time used as termination criterion

Problem size (m x n)	CPU time (in seconds)
5 x 100	1
10 x 100	1
30 x 100	1
5 x 250	5
10 x 250	5
30 x 250	5
5 x 500	10
10 x 500	10
30 x 500	10

Tables 11 and 12 compare the solutions obtained by the ISS approach with those reached by the SS strategy and other operators based on natural selection (TS, RWS, TrS, and SUS). We have tested the performance of the ISS strategy on four test classes varying between low-dimensional MKP instances to high-dimensional MKP instances containing the first ten problems of Chu and Beasley, additionally to the more large instances of Glover and Kochenberger. In Table 11, the first two columns present respectively, the size of instances and their labels, whereas the third reports their corresponding best known solution. The fourth column presents the average solution found by our selection operator for each considered instance over 30 runs, while remaining columns provide for each comparative selection approach the corresponding average solutions. The performance of each method is measured by using the average error criterion introduced by (Beheshti, et al., 2013) and calculated as below:

$$Average\ error = \frac{1}{K} \sum_{i=1}^{K} \frac{b_i - z_i}{b_i} \times 100 \tag{3.8}$$

Where K is the number of used benchmark instances, z_i is the obtained average solution for instance i and b_i represents the best known solution for the i-th instance. The average error is depicted in the penultimate row of each group of instances having the same size in Table 11. In the next row, we list the performance rank of considered algorithms according to their average error. In Table 12, the first three columns present respectively the instance name, the number of items × the knapsack dimensions, and the corresponding best known solution. The fourth

column reports the fixed runtime used to terminate GAs, whereas the remaining columns provide the average solution obtained by the ISS approach and other selection methods for each considered instance over 30 runs. In this table, the best results are marked in bold characters. Note that the default crossover operator used in all these GAs is the uniform crossover.

According to Tables 11 and 12, we remark that the ISS approach outperforms the natural selection methods for low-dimensional MKP instances to high-dimensional MKP instances. Furthermore, by comparing ISS with SS, we can deduce that working in phenotypic space induces a meaningful improvement over the sexual selection strategy whatever the size of the test instances.

Table 11. Performance comparison on different sizes of Chu and Beasely instances ($\alpha = 0{,}25$)

m x n	Benchmark instances	Best Known	ISS	SS	TS	RWS	TrS	SUS
5 x 100	Instance1	24381	23401,9	23423,37	23269	23330,53	23343,43	21023,67
	Instance2	24274	23807,97	23782,3	23591,8	23677,77	23578,17	20627
	Instance3	23551	23169,53	23136,23	23091,87	23084,07	22971,33	19652,53
	Instance4	23534	22552,17	22518,3	22453,63	22501,9	22387,17	20577,7
	Instance5	23991	23491,73	23440,03	23309,63	23360,97	23154,77	20657,53
	Instance6	24613	24113,87	24041,67	23970,5	24043,5	23771,67	20985,93
	Instance7	25591	24890,2	24836,67	24757,47	24731,93	24717,5	20942,83
	Instance8	23410	22756,3	22690,13	22636,97	22689,07	22545,7	20223,93
	Instance9	24216	23543,27	23503,4	23345,43	23463	23479,27	21447,03
	Instance10	24411	23972,07	23960,87	23782,47	23883,43	23743,2	20051,67
Average fitness value error (%)			2,59	2,75	3,21	2,98	3,42	14,76
Performance rank of algorithms			1	2	4	3	5	6
10 x 250	Instance1	117811	113787,5	113562,3	112991,7	113032,3	112721,8	113753
	Instance2	119232	115517,3	115435,8	114951,2	114653,6	114246,4	115381,2
	Instance3	119215	115381,4	115117,3	114333,4	114776	114155,4	115054,4
	Instance4	118813	114920,1	114699	114477,4	114375,5	113868,8	114764,1
	Instance5	116509	112270,4	111996,8	111723,9	111600,5	111179,7	112089,8
	Instance6	119504	115737,8	115564,1	114856,1	115224,2	114366,2	115413,3
	Instance7	119827	115851,5	115773,2	115119,7	115431,7	114294,6	115433,6
	Instance8	118329	114001,1	113827,3	113412,4	113927	112516,4	113834,5
	Instance9	117815	113861,5	113617,6	113301,6	113385,3	112942,8	113532,2
	Instance10	119231	115636,3	115562,3	115030,9	115274,8	114470,9	115573
Average fitness value error (%)			3,32	3,47	3,89	3,76	4,34	3,50
Performance rank of algorithms			1	2	5	4	6	3
30 x 500	Instance1	116056	111247,3	110926,7	110560,2	110611	110260,8	110867
	Instance2	114810	110176,5	110110,8	109618,3	109523,7	108810	109614,1
	Instance3	116712	111848,4	111616,3	111145,1	111262	110467,4	111708,1
	Instance4	115329	110634,3	110510,2	110291,7	109951	109523,7	110308
	Instance5	116525	111670,9	111422,8	110925,3	111058	110578,9	111248,6
	Instance6	115741	111147,1	110820,3	110241,1	110744,8	110194,3	111010,6
	Instance7	114181	109350,2	109101	108434,3	108510,2	107531,9	109118,3

Instance8	114348	109301,7	108996,6	108667,6	108906,7	108160,4	108978,2
Instance9	115419	110544,6	110220,9	109817,3	109877,3	109025,9	110380,8
Instance10	117116	112032,4	111962,2	112036,3	111766,2	110896,8	111779,5
Average fitness value error (%)		4,18	4,37	4,71	4,67	5,26	4,43
Performance rank of algorithms		1	2	5	4	6	3

Table 12. Comparison results on difficult instances of Glover and Kochenberger

Instances	m x n	Best Known	CPU seconds	ISS	SS	TS	RWS	TrS	SUS
gk08	50 x 500	18801	30	**18534,4**	18460,07	18423,93	18437	18392,2	18450,63
gk09	25 x 1500	58085	70	**57025,6**	56879,3	56825,23	56826,5	56711,6	56889,13
gk10	50 x 1500	57292	140	**56879,3**	56295,53	56192,13	56238,53	56122,17	56233,63
gk11	100 x 2500	95231	480	**94024,2**	93724,43	93598,43	93692,77	93534,5	93666

To assess the validity of our conclusions, we apply the Wilcoxon test (Wilcoxon, 1945) for pairwise comparison between the ISS approach and its variants ISS* and ISS**, as well as the SS approach. The Wilcoxon test consists of testing hypotheses about relationships and differences between two correlated samples characterized by an ordinal measurement scale. A null hypothesis assumes that there is no significant difference between the two related samples, while an alternative hypothesis assumes that there is a significant difference between them at a 0.05 significance level. A *p*-value (for probability value) is an asymptotic significance that helps to determine the relationship between the obtained results by comparing it with the significance level. Table 13 summarizes the obtained indicators using average solutions as ordinal-scaled data. Notice that R+ and R- are respectively the sum of ranks of the positive differences (between input data of the two compared approaches) and the sum of ranks of the negative differences. From Table 13, it can be observed that a statistical difference is detected for each compared case with a *p*-value < 0.05. Moreover, the ISS approach is significantly better than the three reference approaches. Its dominance is confirmed by the fact that R+ is significantly larger than R-. It should be emphasized that to apply the Wilcoxon test we have used IBM SPSS Statistic 22.

Table 13. A Wilcoxon test on high-dimensional instances (m = 30 and n = 500)

Algorithm	R+	R-	***p***-Value	Difference
ISS versus SS	465	0	0,000	Yes
ISS versus ISS*	464	1	0,000	Yes
ISS versus ISS**	465	0	0,000	Yes

4.2.2 The results of the proposed two-stage recombination operators

Another experiment is investigated to test the performance of both proposed variants of the two-stage recombination operator (2SR_v1 and 2SR_v2). We have combined 2SR_v1 and 2SR_v2 separately with three different selection strategies, namely the SS strategy (see Table 14), the TS strategy (see Table 15), and the ISS strategy (see Table 16). These tables provide a summary of the average solutions obtained by the two variants 2SR_v1 and 2SR_v2 over a

reduced set containing 30 instances selected among the 270 MKP benchmarks of ORLib dataset and 4 instances among the large and difficult MKP benchmarks of GK dataset. We note that the 30 instances have different tightness ratios since 2SR_v2 is affected by the latter. In Tables 14 and 15, the first three columns present respectively, the instance labels, the tightness ratios, and the corresponding best known solutions. The remaining columns report the average solution found by 2SR_v1 and 2SR_v2 approaches for each instance over 30 runs, as well as those achieved by the comparative crossover operators that are 1PC, 2PC, and UC. As stated previously, the performance of each method is measured by using the average error criterion. In Table 16, the first three columns present respectively the instance names, the number of items × the knapsack dimensions, and their corresponding best known solutions. The fourth column reports the preset time to achieve average solutions, whereas the remaining columns provide the average solutions obtained by 2SR_v1 and 2SR_v2 operators against those obtained by the traditional crossover operators. The best results are marked in bold characters.

From Tables 14-16, we can remark that for all these instances, the two proposed versions of the two-stage recombination operator can reach better results than those obtained by their rival approaches within the same computational time. Moreover, it can clearly be seen that 2SR_v2 is more efficient than 2SR_v1 for low-dimensional MKP instances, but it is outperformed by 2SR_v1 when the size of MKP instances increases.

Table 14. Performance comparison on low-dimensional benchmark instances (m = 30 and n = 100)

m x n	α	Best Known	SS				
			1PC	2PC	UC	2SR_v1	2SR_v2
30 x 100	0,25	21946	20713,2	20785,93	21043,47	21198,6	21628,27
		21716	20271,87	20193,3	20570,1	20441,03	21420,43
		20754	19313,17	19261,23	19623,87	19711,57	20461,13
		21464	20155,3	20155,43	20523,3	20797,47	21201,47
		21844	20310,83	20365,33	20716,47	21071,37	21423,17
	0,5	40767	38354,6	38419,03	39465,5	39689,9	40298
		41308	38890,83	39160,03	39998,87	40375,33	40938,1
		41630	39082,23	39443,07	40240,27	39965,3	41267,27
		41041	38508,7	38714,4	40061,3	40453,57	40687,33
		40889	38511,8	38705,2	39627,13	39863,6	40543,8
	0,75	57494	53421,87	53983,37	56091,77	56895,53	57048,17
		60027	55809,87	56460,43	58541,13	59305,23	59656,1
		58052	53818,43	54406,57	56796,4	57386,63	57661,13
		60776	56145,73	56787,2	59345,9	60158,6	60413,9
		58884	54404,07	54700,4	57256,33	58193,17	58452,4
Averge ftness value error (%)			6,59	6,10	3,45	2,62	1,03
Performance rank of algorithms			5	4	3	2	1

Table 15. Performance comparison on high-dimensional benchmark instances (m=10 and n = 500)

m x n	α	Best Known	TS				
			1PC	2PC	UC	2SR_v1	2SR_v2
10 x 500	0,25	117811	109979,9	110287,5	112991,7	113897,8	115028,5
		119232	111482,2	111654,2	114951,2	117530,4	116306,9
		119215	111653,9	111975,3	114333,4	115770,5	116369,3
		118813	110496,9	111297,1	114477,4	116990,9	115831,1
		116509	108813,8	109033,4	111723,9	112747,3	113763,1
	0,5	217377	198475,3	198740,5	208495,9	214965	207494,7
		219077	197723,3	199595,4	210038,8	214167,4	209051,5
		217847	197292,3	198547,7	208778,3	213760,8	207660,9
		216868	195258,1	196362,1	207921,6	214465,5	206350,6
		213859	193404,5	194842,1	205412,1	209864,3	203988
	0,75	304387	238440,1	242352,8	289083,4	302233,9	295430,2
		302379	234846,9	244022,7	286798	301043	294209,3
		302416	235836,1	246660,9	287287,6	300114,6	294430,9
		300757	236304,7	241980,5	285448,2	299451,5	292087,3
		304374	239420,1	247039,6	289174,9	303228	295372,5
Averge ftness value error (%)			12,62	11,51	4,35	1,55	3,30
Performance rank of algorithms			5	4	3	1	2

Table 16. Results for some difficult instances of Glover and Kochenberger

Problem Instances	m x n	Best Known	CPU seconds	ISS				
				1PC	2PC	UC	2SR_v1	2SR_v2
gk08	50 x 500	18801	30	18320,97	18342,94	18534,4	**18586,9**	18391,83
gk09	25 x 1500	58085	70	56296,27	56327,44	57025,6	**57831**	56284,13
gk10	50 x 1500	57292	140	55881,3	55933,66	56780,87	**56879,3**	55843,7
gk11	100 x 2500	95231	480	93332,47	93381,77	94024,2	**94495,7**	93254,1

In order to further evaluate our ISGA algorithm, we compared it with two GAs that combine the SS strategy with 2SR_v1 and 2SR_v2 separately. Thus, a Friedman test[21] is carried out for testing the difference between the comparative algorithms. We note that we have also used IBM SPSS Statistic 22 to apply the Friedman test. Table 17 records the results of Friedman test performing on average solutions. Thus, the resulting p-value is 0.075 which clearly indicates that there is no significant statistical difference between a GA that uses ISS with 2SR_v1 and the one that combines ISS with 2SR_v2. Moreover, The Friedman test reveals that the differences lie in the pair of GAs using ISS with 2SR_v1 and SS with 2SR_v1, as well as the pair of GAs using ISS with 2SR_v2 and SS with 2SR_v2. It can also be observed that the GAs using ISS with 2SR_v1 and ISS with 2SR_v2 are respectively superior to those using SS with

[21] The Friedman test is a non-parametric test based on ranks, having a principle similar to that of Wilcoxon test, except that it is especially designed for three or more repeated measurements of ordinal data (for more details see (Friedman, 1940))

2SR_v1 and SS with 2SR_v2 since the sum of their positive ranks is significantly larger than the sum of their negative ranks.

Table 17. A Friedman test on some benchmark instances (m = 30 and n = 250)

Algorithms	R+	R-	p-Value	Difference
ISS with 2SR_v2 versus ISS with 2SR_v1	319	146	0,075	No
ISS with 2SR_v1 versus SS with 2SR_v1	390	75	0,001	Yes
ISS with 2SR_v2 versus SS with 2SR_v2	401	64	0,001	Yes

5 Conclusion

In this chapter, we proposed a new sexual selection strategy inspired from that of (Varnamkhasti & Lee, 2012a), as well as two variants of the two-stage recombination operator proposed by (Aghezzaf & Naimi, 2009). These genetic operators are combined in the same GA, called ISGA, to improve its performance in exploring and exploiting the search space. The proposed operators are tested on a wide set of 0/1 MKP benchmark instances and the results are compared with those obtained by several GAs using traditional methods. The evaluation of ISGA was based on some robust criteria including the percentage gap, the average error, in addition to non-parametric statistical tests: Friedman test and Wilcoxon test to detect statistical differences between the obtained results.

The experiments demonstrated that our algorithm could produce solutions of good quality while fixing a priori two stopping criteria: A maximum number of generations and a fixed amount of time. In the first case, our algorithm yields good solutions in a reasonable runtime. In the second case, the results confirm that our algorithm is still more efficient than the comparative algorithms despite fixing the same and relatively high execution time for all compared algorithms in order to give them more chance to improve their results. As future work, we plan to adapt the proposed operators to other variants of 0/1 MKP as well as other binary combinatorial optimization problems. In addition, another enhancement may be related to the steps of selecting male chromosomes in order to speed up the search process.

Chapter 4: A genetic algorithm based on *k*-means method for solving 0/1 MKP

1 Introduction

For decades, researchers have attempted to develop algorithms in optimization satisfying superior performance. There exist two most preferred approaches by the researchers to improve an existing algorithm: hybridization and modification. The former is the process of mixing of at least two distinct algorithms, while the latter is the process of modifying some mechanisms in order to improve the search ability. The hybridization can be realized between heuristic approach/exact approach, exact approach/exact approach or heuristic approach/ heuristic approach. These hybrids usually reach good results because they can simultaneously exploit the advantages of both types of methods. A simple hybridization approach is combining an existing algorithm with a genetic algorithm (GA). Such a combination has received significant interest in recent years and is being increasingly used to solve real-world problems. A genetic algorithm is able to incorporate other techniques within its framework to produce a hybrid that reaps the best from the combination. On the other hand, it can be integrated within many existing algorithms as a complementary tool for improving their search ability.

In this chapter, we propose a GA algorithm that uses a clustering algorithm to organize the population and select the parents for recombination. A clustering algorithm aims to organize data into sensible groupings (clusters) according to measured or apparent similarities. Clustering has been successfully integrated into GAs to solve clustering problems facing decision makers in different domains such as biology, medicine, machine learning, pattern recognition, image analysis, and data mining. However, scarce studies that have used such hybridization in optimization problems. The performance of our proposed hybrid GA is investigated on 0/1 MKP benchmark instances. The remainder of this chapter is organized as follows. Section 2 outlines the principle of the used clustering algorithm that is the *k*-means method. In the next section, different forms of combinations of genetic algorithms with other optimization techniques for solving 0/1 MKP, are reviewed. All operators of the proposed hybrid GA are detailed in section 4. The computational results and comparison of our GA approach with other classical GAs, are carried out in section 5. The conclusions are giving in section 6.

2 Clustering methods

A clustering algorithm (Berkhin, 2006) is an important unsupervised classification technique that consists of classifying data into subsets, called “clusters”, having some features in the context of a specific problem. In fact, data are grouped into clusters in such a way those belonging to the same cluster should be similar in some sense, while those coming from different clusters should be dissimilar in the same sense. A similarity metric should be used to measure the similarity of data, which is defined according to the nature of data and the purpose of analysis. Some examples of useful measure techniques for quantitative features are: Minkowski distance, Euclidian distance, Manhattan distance (see chapter 3), and Tchebychev

distance that is also a special case of Minkowski distance when $p \rightsquigarrow \infty$, for more metrics we invite the readers to refer to the work of (Xu & Wunsch., 2005).

There are numerous clustering algorithms that can be broadly categorized into hierarchical and partitional algorithms. Hierarchical clustering groups data in the form of a sequence of partitions, either from singleton clusters to a cluster including all individuals or vice versa, whereas partitional clustering directly divides data into some prespecified number of clusters without imposing a hierarchical structure. The *k*-means algorithm (MacQueen, 1967) is a well-known example of clustering methods, which obtains *k* partitions of data through an iterative refinement such that the similarity distance between data and their cluster centroids is minimized. The *k*-means algorithm starts with *k* initial centroids that are randomly chosen and assigns each data to a cluster having the nearest centroid. Then, the centroid of each obtained cluster is recomputed and data are reassigned to the nearest new centroids. This process is repeated until the centroid of each cluster remains unchanged. The pseudo code of *k*-means algorithm is presented in Algorithm 6:

1.	**Input*: k* is the number of clusters and *D* is a set of objects**
2.	Choose randomly *k* objects from *D* as initial cluster centroids
3.	Repeat
4.	Assign each object to the cluster having the closest centroid
5.	Recalculate new centroids for generated clusters
6.	Until no change
7.	**Output: *k* clusters**

Algorithm 6.The pseudo code of *k*-means algorithm

Despite its popularity, *k*-means algorithm has the disadvantage that it is highly sensitive to the random choice of initial centroids that causes the algorithm to be falling into local optimum (see a relevant review of (Jain, et al., 1999)). Several variants of the *k*-means algorithm have been reported in the literature. Some of them attempt to select a good initial partition so that the algorithm is more likely to find global minima ((Likas, et al., 2003) and (Arthur & Vassilvitskii, 2007)). Another variation is to hybridize GA with *k*-means algorithm in order to provide a certain perturbation to escape from local minima (Bandyopadhyay & Maulik, 2002). In contrast, there are scarce studies that have integrated *k*-means into GA in such a way the resulting hybrid approach can tackle optimization problems.

3 Hybrid genetic algorithms for solving 0/1 MKP

There are many approaches that combine genetic algorithms with other methods to solve 0/1 MKP. Some of these hybrid approaches mainly aim to improve the quality of solutions by providing equilibrium between intensification and diversification strategies. However, other algorithms originally devoted to continuous optimization problems, incorporate the genetic operators as a technique of binarization. Both categories take profit from the straightforward implementation of the GA, its ability to be adapted to any kind of optimization problems,

additionally to its effectiveness for solving 0/1 MKP and consequently for other complex optimization problems.

3.1 GA with exact method

The solutions obtained by relaxing 0/1 MKP can be used to derive promising initial solutions for a subsequent algorithm, on one hand. On the other hand, they can be integrated into other algorithms to solve heuristically 0/1 MKP as mentioned in chapter 2. The approaches proposed by (Chu & Beasley, 1998), (Raidl, 1998), and (Deane & Agarwal, 2012) are good examples of hybrid genetic algorithms in which the solution of the relaxed 0/1 MKP was exploited. In addition to (Gallardo, et al., 2005) that have integrated the branch and bound method within GA to cooperate together in order to exchange information. Their proposed approach can provide high-quality solutions better than that achieved by the GA and the Branch and Bound separately.

3.2 GA with heuristic method

To mention just a few, an example of cooperation between a heuristic and a GA is the hybrid genetic algorithm proposed by (Chu & Beasley, 1998), in which they have incorporated a constructive heuristic within their proposed repair operator. Besides, (Raidl, 1999) that has proposed a GA working in phenotype space in which he has incorporated separately two new heuristics to decode chromosomes. The two resulting GAs perform better than the proposed heuristics when they are applied to 0/1 MKP directly.

3.3 GA with metaheuristic method

On the flip side, a wide range of papers about cooperation between GAs and other metaheuristics are published. (Rezoug, et al., 2015) have proposed two different hybrid approaches. The first one consists of combining GA with stochastic local search, whilst the second one is a combination of GA with simulated annealing. The experimental results of both approaches seem to be competitive in solving 0/1 MKP. (Deane & Agarwal, 2012) have hybridized a new GA with a new neural approach. The experiments showed that such a combination provides better solutions in comparison with the GA and the neural approach separately. In the same way, (Bole & Kumar, 2017) have proposed a hybrid algorithm that combines the tournament selection of GA with bee's decision-making process. The computational experiments demonstrated the robustness and accuracy of such hybridization against GA, PSO, and glowworm swarm optimization. Furthermore, (Abdel-Basset, et al., 2018) have incorporated a crossover operator in the flower pollination optimization algorithm. The simulations showed that their hybrid approach yields better solutions in shorter runtime against other similar algorithms recently stated in the literature. On the other hand, (Peng, et al., 2016) have proposed a new binary differential evolution algorithm (DEA) in which

dichotomous mechanisms[22] are fused into crossover and mutation operations. The binary approach was tested on the 0/1 knapsack problem and 0/1 MKP instances proving its efficiency when compared with two variants of PSO and three variants of binary DEA. (Gherboudj, et al., 2012) have developed a new hybrid binary PSO by combining some features of PSO and crossover operation of GA. The computational results showed that this hybrid yields good and promising solution quality in comparison with another version of PSO and a new cuckoo search algorithm (Gandomi, et al., 2013).

3.4 GA with clustering method

To the best of our knowledge, there are quite rare papers that are interested in making cooperation between GAs and clustering methods to deal with 0/1 MKP. We cite the work of (Gupta, et al., 2017) as an example of such hybridization. They have proposed a clustered GA to solve 0/1 MKP, wherein they have used a roulette wheel selection with fuzzy techniques to create a mating pool, then they have applied hierarchical clustering method to form clusters from which they select pairs of parents for the crossover operation. According to simulations carried out on 30 instances of 0/1 MKP, the hybrid GA performs favorably against the traditional GA.

4 The proposed hybrid approach

In this section, we describe our proposed GA that uses a selection operator based on *k*-means method, named GA-based *k*-means method. The motivation behind this hybridization is to provide a high level of diversification at the start of the algorithm in order to avoid premature convergence. For this reason, we incorporate *k*-means algorithm into GA, especially, in the selection strategy. By fixing *k* to 2 a priori, *k*-means creates two clusters of chromosomes. Hence, the objective is to select pairs of parents having dissimilar features, one from the first cluster and the second one from the other cluster. Such a selection gives a chance to unfit chromosomes to participate in reproduction hoping to obtain fitter offspring chromosomes in the upcoming generations. To go even further, this selection based on *k*-means techniques will push the genetic search toward diversification, while the crossover operation will push the search toward intensification.

It should be mentioned that the choice of *k*-means as a clustering algorithm rests on its straightforward process, its ability to work on quantitative features, as well as its efficiency when the size of data (*i.e.* the number of items in the context of 0/1 MKP) is large-scaled (see the work of (Mann & Kaur, 2013)).

[22]The dichotomous mechanism is inspired from dichotomous thinking of psychology, which means the tendency of only seeing extremes, also known as “black and white thinking”.

4.1 Encoding schema

To use *k*-means algorithm in GA, we treated the population of chromosomes in phenotype space. A chromosome represents the multidimensional knapsack, whereas genes store the information about items. This information is determined by a gene fitness function that is calculated as following:

$$f(i) = \frac{p_i}{\sum_{j=1}^{m} w_{ij}} \quad ; \quad \forall\, i = 1, \ldots, n \tag{4.1}$$

Hence, the chromosomes are encoded as below:

Table 18. Encoding schema of chromosomes

1	2	3	…	n-1	n
$\frac{p_1}{\sum_{j=1}^{m} w_{1j}}$	0	$\frac{p_3}{\sum_{j=1}^{m} w_{3j}}$	…	$\frac{p_{n-1}}{\sum_{j=1}^{m} w_{(n-1)j}}$	0

The first row reports the index of items, while the second presents the corresponding genes' value. The gene value, in phenotype space, equals the gene's fitness value if the corresponding item is selected; and it equals 0 otherwise. As a reminder, the length of chromosomes corresponds to the number of available items. The reason why we adopt this representation is due to the fact that it is direct and it makes it easy to measure the clustering metric of *k*-means algorithm.

4.2 Initial population

The chromosomes of the initial population are randomly generated. However, the random allocation of genes into chromosomes will end up with some infeasible chromosomes. That is why we have applied a repair and improvement operator (Zouache, et al., 2016) which seems to be suitable for our encoding schema. This repair operator performs as following: Firstly, the items having the lowest ratio $p_i / \sum_{j=1}^{m} w_{ij}$ (or gene fitness function in our context) are removed iteratively until the capacity constraints will be respected ($\sum_{i=1}^{n} w_{ij} x_i \leq c_j,\ \forall\, j \in \{1, \ldots, m\}$). Secondly, the items having the highest ratio are added in turn until at least one of capacity constraints is violated.

4.3 The proposed selection strategy

Based on *k*-means principle (*k* is fixed to 2), the population is divided into two groups so that the chromosomes belonging to the same group are closely similar and the ones coming from separate groups are distant. We have used as a similarity metric the Manhattan distance (MD) that should be minimized as defined below:

$$MD(X, Y_k) = \sum_{i=1}^{N} |gene_i(X) - gene_i(Y_k)|\,; \quad for\ k = 1,2 \tag{4.2}$$

Where $X = \{X_l\},\ l = 1, \ldots, (population\ size - 2)$, $gene_i(X)$ is the value of the i-th gene in the chromosome X, and $gene_i(Y_k)$ is the value of the i-th gene in the centroid Y_k of cluster k. It is emphasized that the centroid is a chromosome with a specific characteristic. The initial centroid of the first cluster is the fittest chromosome in the whole population, while the initial centroid of the second cluster is the chromosome having the lowest fitness value. For each cluster k, the genes' value of the new centroid Y_k^* is computed as:

$$gene_i\,(Y_k^*) = \frac{\sum_{l=1}^{\Omega_k} f_l(i)}{\Omega_k}; \quad for\ i = 1, \ldots, n \tag{4.3}$$

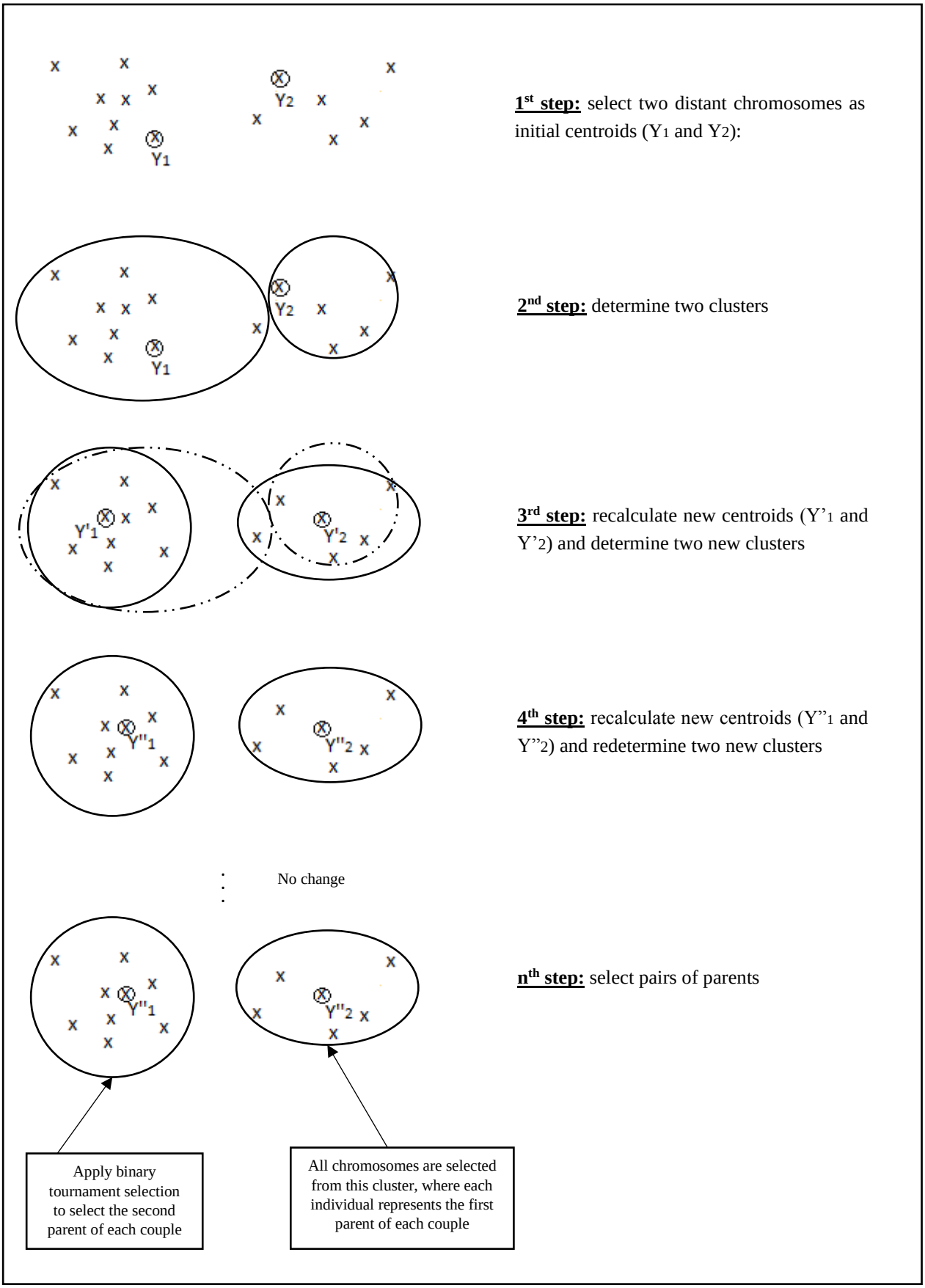
Where Ω_k is the cardinal of cluster k and $f_l(i)$ is the fitness value of the i-th gene in chromosome l.

After partitioning the population in two clusters. All chromosomes of the cluster having the lowest cardinal could participate in the reproduction phase regardless of their fitness value. They are selected in a sequential fashion without replacement, which means that each one will be selected once for mating. Then, we choose a mate for each chromosome by applying the binary tournament selection in the next cluster. In fact, two chromosomes are randomly chosen, the candidates' fitness values are then compared, and the chromosome of the highest fitness becomes the mate of the current chromosome. This mate selection favors the fittest chromosomes among the cluster, which pushes the search towards the intensification strategy. By this way, the selection scheme provides a balance between intensification and diversification strategies in the search process at an early stage of GA. The main steps of the proposed selection operator are shown in Figure 14.

4.4 The variation operators and replacement strategy

The variation operators include the crossover and mutation operators. Crossover operation in the GA-based *k*-means method is accomplished via the uniform crossover operator. Once crossover operation is executed, some of the offspring chromosomes undergo a mutation operation with a preset probability level. For this purpose, we have adopted the so-called bit-flip mutation as a default strategy.

In order to initialize the population of the next generation, we have made use of the generational replacement strategy. The procedure is stopped if a maximum number of generations is reached. The main steps of the GA-based *k*-means method are depicted above in Figure 15.



1st step: select two distant chromosomes as initial centroids (Y_1 and Y_2):

2nd step: determine two clusters

3rd step: recalculate new centroids (Y'_1 and Y'_2) and determine two new clusters

4th step: recalculate new centroids (Y''_1 and Y''_2) and redetermine two new clusters

nth step: select pairs of parents

Figure 14. The main steps of the proposed selection strategy

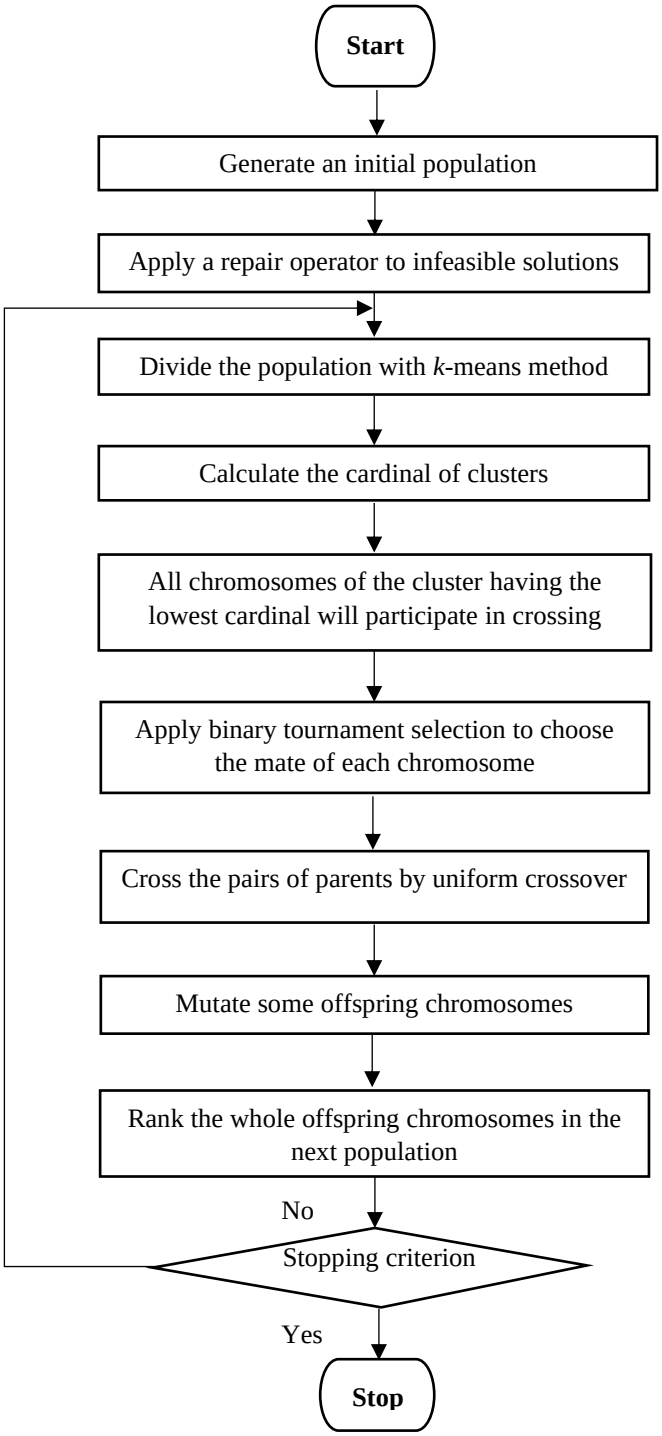


Figure 15. The flowchart of the GA-based *k*-means method

5 Experimental results

5.1 Experimental design

To investigate how GA-based on *k*-means method performs, we have compared it to GAs using traditional selection strategies, namely the binary Tournament Selection (TS), the Roulette Wheel Selection (RWS), the Truncation Selection (TrS), and the Stochastic Universal Selection (SUS). These algorithms have been implemented in JAVA and executed on a PC with Intel® Core™ i5, 2.5 GHz and 4.0 Go of RAM, running on 64-bits Windows 7 operating

system. Before presenting the results, we report in Table 19 the parameters used in our GA as well as the comparative GAs.

Table 19. Parameters setting

Parameters	Value
Population size	100
Crossover rate	0.50
Mutation rate	1/Number of items
Maximum generation	1500
Truncation threshold	50%
Tournament size	2

The performance of the proposed GA is evaluated on the benchmark instances proposed by Glover and Kochenberger, see chapter 3 - section 4 for more details. The quality of solutions of each algorithm is measured by the average fitness across 30 runs and the percentage gap between the achieved solutions and the best benchmark solutions. On the other hand, the computational time is measured by the average value of time required to find the final solution.

5.2 Experimental results

Table 20 outlines the average fitness of our GA against that of conventional GAs. The first column presents 11 problem instances. The second column shows, respectively, the number of items n and the number of knapsack dimensions m. The third column reports the best benchmark solutions, while the empirical results of all algorithms are shown in the remaining columns. As stated earlier, the comparison of solution quality is performed based on two performance measures: The average fitness presented in the first row of each instance and the percentage gap reported in the second row. The computing time (in seconds) is depicted separately in Figure 16.

Table 20. Comparison of the average fitness value obtained by GA- based *k*-means and the comparative methods (the best results are in bold).

Instances	*n* x *m*	Best known		kMS	TS	RWS	TrS	SUS
gk01	100 x 15	3766	Avg	**3690,20**	3681,30	3688,40	3673,93	3686,17
			Gap %	2,01	2,25	2,06	2,44	2,12
gk02	100 x 25	3958	Avg	3875,53	3869,27	**3879,03**	3860,17	3877,07
			Gap %	2,08	2,24	2,00	2,47	2,04
gk03	150 x 25	5650	Avg	**5540,40**	5526,37	5533,47	5522,97	5532,77
			Gap %	1,94	2,19	2,06	2,25	2,07
gk04	150 x 50	5764	Avg	**5656,63**	5632,23	5639,23	5624,87	5640,07
			Gap %	1,86	2,29	2,16	2,41	2,15
gk05	200 x 25	7557	Avg	**7417,20**	7389,23	7405,80	7390,57	7409,77
			Gap %	1,85	2,22	2,00	2,20	1,95
gk06	200 x 50	7672	Avg	**7524,40**	7515,87	7519,57	7494,20	7517,40
			Gap %	1,92	2,04	1,99	2,32	2,02

gk07	500 x 25	19215	Avg	**18849,20**	18813,97	18832,97	18755,20	18831,93
			Gap %	1,90	2,09	1,99	2,39	1,99
gk08	500 x 50	18801	Avg	**18458,87**	18426,40	18430,83	18434,80	18399,80
			Gap %	1,82	1,99	1,97	1,95	2,13
gk09	1500 x 25	58085	Avg	**56891,80**	56815,43	56762,90	56758,53	56872,83
			Gap %	2,05	2,19	2,28	2,28	2,09
gk10	1500 x 50	57292	Avg	56256,50	56158,03	**56282,13**	56133,77	56252,30
			Gap %	1,81	1,98	1,76	2,02	1,81
gk11	2500 x 100	95231	Avg	**93664,40**	93604,20	93641,53	93584,40	93598,00
			Gap %	1,65	1,71	1,67	1,73	1,71

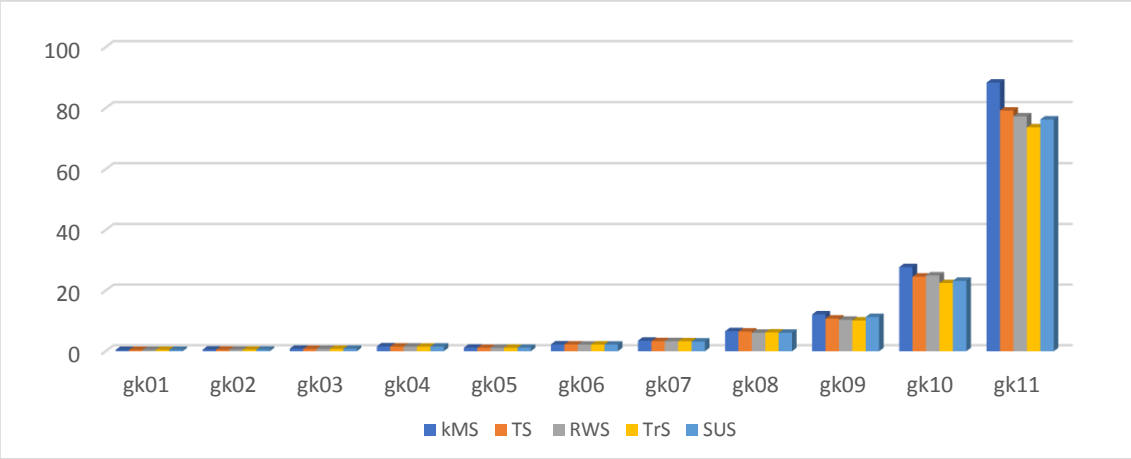


Figure 16. A comparison of average time

It is worthwhile mentioning that all algorithms have used the same parameters. As shown in Table 20, the proposed GA based on *k*-means selection strategy (denoted by kMS) has in general the best average fitness in comparison to GAs using other selection processes, except for the instances gk02 and gk10 in which the GA using RWS performs better than our hybrid approach. On the other hand, for the most considered test instances, the proposed GA has a computational time comparable to that consumed by the other algorithms. However, for the very large-scale instances, there is a little more meaningful difference between our algorithm and the other algorithms in terms of convergence speed.

6 Conclusion

This chapter proposed a new selection strategy based on *k*-means clustering method, which is independent from the structure of the problem at hand. It is based on two main steps: It applies firstly *k*-means method to preform *k* (= 2) clusters, then it selects from each cluster one parent. The first parent is selected in a random way, whilst the second parent is selected according to its fitness value. The fact of selecting parents from different clusters brings forth non-identical offspring chromosomes, thereby preventing premature convergence to local optima. To examine the performance of our proposed genetic algorithm, we solved some 0/1 MKP benchmark tests that include so large-scaled instances. In addition, we compared our

algorithm with other genetic algorithms using traditional selection operators. The experimental results showed that the proposed selection strategy improves the quality of achieved solutions.

Future research could test the performance of a genetic algorithm using this proposed selection strategy and the so-called two-stage recombination operator proposed by (Aghezzaf & Naimi, 2009) or its variants (2SR_v1 or 2SR_v2). Such a combination seems to be promising and synergetic since it would take profit from dissimilarity between parents to outbreed non-identical offspring chromosomes, and hence increasing genetic diversity in the population.

Part II: The two-dimensional bin packing problem

Chapter 5: Definition, variants, and resolution approaches

1 Introduction

As a reminder, when the number of containers is large enough to pack all available items, the objective function consists of minimizing the number of used containers. An example of such an optimization problem is the so-called bin packing problem. In this chapter, we mainly consider a generalizing case of this latter, which is the two-dimensional bin packing problem (2D-BPP).

In 2D-BPP, we are giving a set of n rectangular items, each has a width and height; and an unlimited number of identical rectangular objects called bins, and having larger dimensions than those of items. The problem consists of allocating orthogonally all items to a minimum number of bins without overlapping. The items packed in each bin should respect its loading capacity. The bin packing problem, particularly the 2D-BPP, is one of the most studied problems after the knapsack problem, in the so-called Cutting & Packing problems category ((Dyckhoff, 1990), (Wäscher, et al., 2007)). This problem is an NP-hard combinatorial optimization problem in the strong sense since it is a generalization of the one-dimensional bin packing problem (1D-BPP) in which only one dimension is considered (width or height). This problem has been discussed for many years, and it has been applied to several fields: In industry (Hopper & Turton, 1999) when rectangular figures of wood, glass, or paper have to be cut, in transportation sector (Leung, et al., 2011), computer networking field (Stille, 2008), and so on.

This chapter surveys the studies interested in the 2D-BPP. It is organized as following. In section 2, we define the problem by giving some of its mathematical models. Several variants and some of the real-world applications are highlighted in section 3. We proceed then with the definition of lower bounds in section 4, while sections 5, 6, and 7 are dedicated respectively to exact methods, heuristic, and metaheuristics. In section 8, we close briefly this chapter by some conclusions.

2 Definition and mathematical formulations

2.1 Definition

Formally, given a set of n small rectangles, called "items", where each item j is characterized by a width w_j and a height h_j $(j \in \{1, \ldots, n\})$, and an unlimited number of large rectangles, called "bins", having a fixed width W and height H. The goal is to minimize the number of used bins to pack all items while respecting their loading capacities $(W \times H)$ (see Figures 17 and 18). The items must respect orthogonal packing (*i.e.* the items might be packed with their edges parallel to those of bins). Besides, each item should be packed in one and only one bin. It is assumed, without loss of generality, that all input data are positive integers satisfying $h_j \leq H$ and $w_j \leq W$, for all $j \in \{1, \ldots, n\}$.

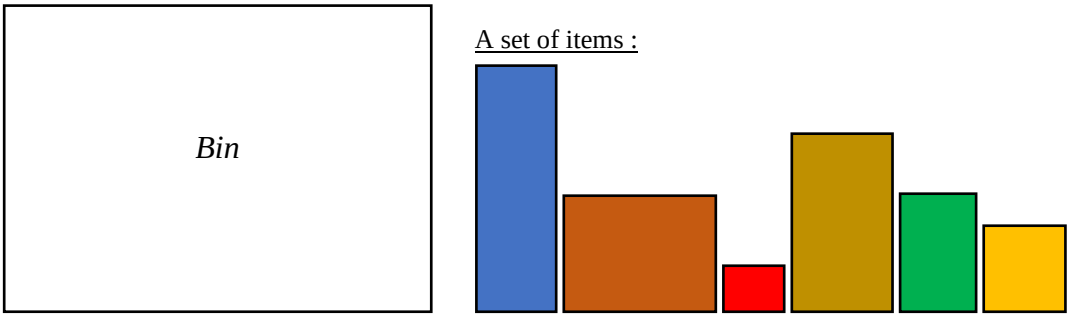


Figure 17. An example of a 2D-BPP instance

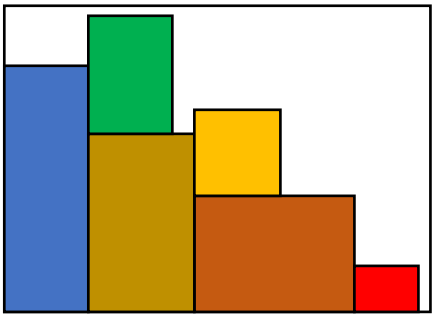


Figure 18. A possible solution of the above instance

2.2 Mathematical models

The first mathematical formulation of 2D-BPP is introduced by (Gilmore & Gomory, 1965) that is considered as an extension of their mathematical formulation devoted to 1D-BPP ((Gilmore & Gomory, 1961), (Gilmore & Gomory, 1963)). It consists of enumerating all subsets of items, called "patterns", which can be packed into a single bin. Each pattern j ($\in \{1, \dots, m\}$) is represented by a binary vector $V_j = (v_{ij})$, where $i \in \{1, \dots, n\}$, so that :

$$v_{ij} = \begin{cases} 1 & if\ the\ item\ i\ belongs\ to\ the\ pattern\ j \\ 0 & otherwise \end{cases} \quad (5.1)$$

Thus, the model of 2D-BPP is described below:

$$\begin{cases} Min \sum_{j=1}^{m} x_j & (5.2) \\ s.t \sum_{j=1}^{m} v_{ij}\, x_j = 1\,; \quad \forall\, i = 1, \dots, n\,; & (5.3) \\ \sum_{i=1}^{n} v_{ij}\, h_i \leq H\,; \quad \forall\, j = 1, \dots, m\,; & (5.4) \\ x_j \in \{0,1\}\,; \quad \forall\, j = 1, \dots, m; & (5.5) \end{cases}$$

Equation (5.2) aims to minimize the number of possible patterns. Equation (5.3) ensures that each item belongs to only one pattern. Equation (5.5) represents a binary decision variable that takes the value 1 if pattern j represents a solution of 2D-BPP and the value 0 otherwise. It is

observed that (5.2), (5.3) and (5.5) are valid equations for 1D-BPP. However, the only difference being that each V_j ($\forall\, j = 1, \dots, m$) must respect the second dimension H of each bin (see equation (5.4)), knowing that the dimension W is already considered in 1D-BPP model.

Another mathematical formulation is developed by (Fekete & Schepers, 2004) based on graph theory concepts. Let $G_w = (V, E_w)$ and $G_H = (V, E_H)$ be two graphs having a vertex v_i associated with each item i, and an edge between two vertices (v_i, v_j) if and only if the projections of items i and j on the horizontal (resp. vertical) axis overlap (see Figure 19). According to Fekete and Schepers, a feasible solution should meet the following conditions:

1. For each set of solutions S of G_w (resp. G_H), $\sum_{v_i \in S} w_i \leq W$ (resp. $\sum_{v_i \in S} h_i \leq H$).
2. $E_w \cap E_h = \emptyset$

The first condition means that the items hold into bins, whereas the second indicates that the items must not overlap.

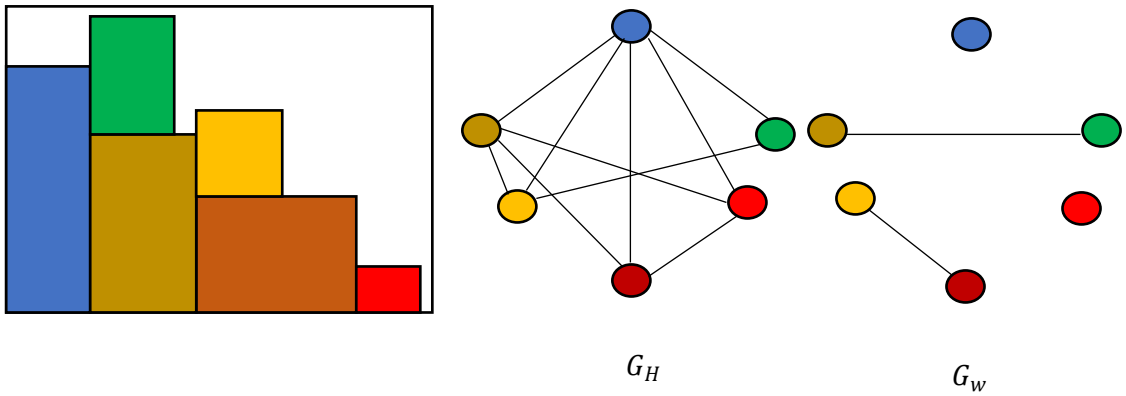


Figure 19. Fekete and Schepers's modeling approach

2.3 Variants and extensions of 2D-BPP

Many real-life problems can be modeled as 2D-BPP. Each real problem aims to a specific objective function and imposes some practical constraints, which are in most cases related to the orientation of items, guillotine cuts (in the context of cutting problems), and conflict between items.

2.3.1 Two dimensional strip packing problem

An important variant of 2D-BPP is the two-dimensional (2D) strip packing problem, in which the items have to be packed in an open-ended strip of fixed width and infinite height, so as to minimize the total height of the packing. Let $I = \{1, \dots, n\}$ be a set of n rectangles. Each rectangle is characterized by a width w_i and a height h_i, while the position of each rectangle i in the strip is represented by the coordinate (x_i, y_i). The 2D strip packing problem is formulated as following:

$$\begin{cases} Min\ H & (5.6) \\ s.t\ \ x_i + w_i \leq \mathrm{W}\,; \quad \forall\, i \in I & (5.7) \\ \quad y_i + h_i \leq \mathrm{H}\,; \quad \forall\, i \in I & (5.8) \\ \quad x_i + w_i \leq x_j \text{ or } x_j + w_j \leq x_i \text{ or} \\ y_i + h_i \leq y_j \text{ or } y_j + h_j \leq y_i; \quad \forall\, i,j \in I; i \neq j & (5.9) \\ \quad x_i, y_i \geq 0\,; \quad \forall\, i \in I & (5.10) \end{cases}$$

Equations (5.7), (5.8), and (5.10) require that all rectangles are within the strip characterized by a width W and an infinite height H, while equation (5.9) prevents rectangles from overlapping. For more detail (see the work of (Kenmochi, et al., 2009)).

2.3.2 2D-BPP with rotation

The items may either have a fixed orientation or they can be rotated by 90°. This possibility of rotation or not gives rise to the so-called 2D-BPP with rotation (see Figure 20). Although the rotation is possible in the most packing applications and the case of plain materials, it is not allowed in some cutting problems (*e.g.* when the items to be cut are decorated or corrugated).

(a) A possible solution of 2D-BPP with fixed orientation (b) A possible solution of 2D-BPP with rotation

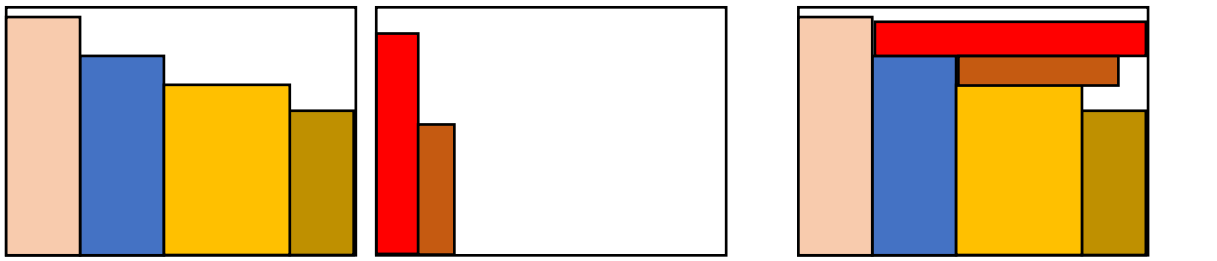


Figure 20. 2D-BPP with rotation

2.3.3 2D-BPP with guillotine cuts

Another variant appears in cutting problems when the guillotine cuts are required, *i.e.* the items are obtained through a sequence of edge-to-edge cuts parallel to the edges of the bin (see Figure 21). The guillotine constraint is frequently imposed in cutting problems due to technological characteristics of automated cutting machines, whereas it is generally not imposed in packing context.

Guillotine cutting is respected Guillotine cutting is not respected

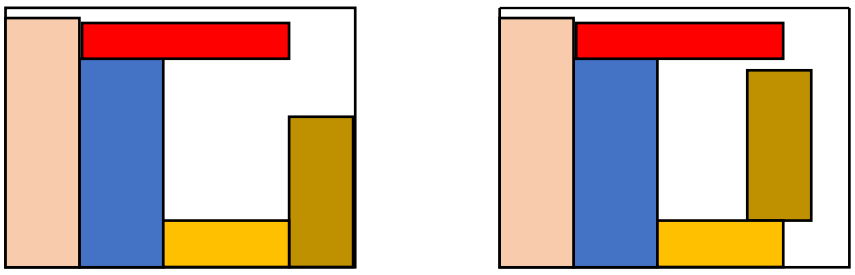


Figure 21. 2D-BPP with guillotine cuts

It should be notated that four other variants result from the combination of the two indicated constraints:

- 2D-BPP with guillotine cuts and fixed orientation.
- 2D-BPP with guillotine cuts and the possibility to rotate items by 90°.
- 2D-BPP with free cutting and fixed orientation.
- 2D-BPP with free cutting and the possibility to rotate items by 90°.

Obviously, a feasible solution of 2D-BPP with guillotine cuts and fixed orientation is feasible of the three other variants of 2D-BPP. Besides, a feasible solution of 2D-BPP with guillotine cuts and the possibility to rotate items by 90°, as well as 2D-BPP with free cutting and fixed orientation, is also feasible of 2D-BPP with free cutting and the possibility to rotate items by 90°. The reader is referred to the work of (Lodi, et al., 1999) for more details.

Notice that in our considered problem no further restriction is taken into account: The orientation of items is free and the guillotine cutting is not imposed.

2.3.4 2D-BPP with conflicts

Let $I = \{1, \dots, n\}$ be a set of rectangular items, each is characterized by a width w_i and a height h_i. Given an unlimited number of rectangular bins and a conflict graph $G = (I, E)$, where $(i, j) \in E$ if items *i* and *j* are in conflict. The aim of the 2D-BPP with conflict is to minimize the number of bins used to pack all items of I, provided that the content of any bin consists of compatible items fitting within the bin capacity without overlapping. The mathematical formulation of 2D-BPP with conflict is proposed by (Khanafer, et al., 2012) as following:

$$
\begin{cases}
Min \sum_{k=1}^{n} z_k & (5.11)\\
s.t \sum_{k=1}^{n} \delta_{ik} = 1\,; \quad \forall\, i \in I\,; & (5.12)\\
\delta_{ik} \le z_k\,; \quad \forall\, i \in I \text{ and } \forall\, k = 1, \dots, n\,; & (5.13)\\
l_{ij} + l_{ji} + b_{ij} + b_{ji} \ge 1 - (1 - \delta_{ik}) - (1 - \delta_{jk}); \quad \forall\, i, j \in I \text{ and } \forall\, k = 1, \dots, n & (5.14)\\
x_i + w_i \le x_j + W \times (1 - l_{ij}); \quad \forall\, i, j \in I & (5.15)\\
y_i + h_i \le y_j + H \times (1 - b_{ij}); \ \forall\, i, j \in I & (5.16)\\
x_i \le W - w_i; \ \forall\, i \in I & (5.17)\\
y_i \le H - h_i; \ \forall\, i \in I & (5.18)\\
\delta_{ik} + \delta_{jk} \le 1; \ \forall (i, j) \in E \text{ and } \forall\, k = 1, \dots, n & (5.19)\\
l_{ij}\,, b_{ij} \in \{0,1\}\,; \quad \forall\, i, j \in I & (5.20)\\
\delta_{ik} \in \{0,1\}; \ \forall\, i \in I \text{ and } \forall\, k = 1, \dots, n\,; & (5.21)\\
z_k \in \{0,1\}; \ \forall\, k = 1, \dots, n\,; & (5.22)\\
x_i, y_i \in \mathbb{N}\,; \ \forall\, i \in I & (5.23)
\end{cases}
$$

Equation (5.12) ensures that each item i is placed at most in one bin. Equation (5.13) indicates that each bin k is used if at least one item is packed in it. Equation (5.14) represents the constraint of no-overlapping by ensuring that two items have to be one above the other or one on the left to the other. Equations (5.15) and (5.16) link the coordinates of items with the variables b_{ij} and l_{ij}. Equations (5.17) and (5.18) indicate that items must respect the capacity of each used bin. Equation (5.19) ensures that two conflicting items cannot be packed in the same bin. Equations (5.20), (5.21), (5.22), and (5.23) represent the binary decision variables of the problem: z_k is equal to 1 if the bin k is used and 0 otherwise, the variable l_{ij} is equal to 1 whether the item i is located left to j and 0 otherwise, similarly b_{ij} is equal to 1 whether the item i is located below j and 0 otherwise, while the variable δ_{ik} is equal to 1 if the item i is packed into bin k and 0 otherwise. Finally, (x_i, y_i) is the lower left coordinates of the i-th item.

In some fields, instead of packing conflicting items totally in different bins, they are just separated by a safety distance in the same bin as it is described in the paper of (Hamdi-Dhaoui, et al., 2012).

2.3.5 Two-dimensional vector bin packing problem

The two-dimensional vector packing problem is an extension of 2D-BPP. It consists of assigning n items to a certain number of available bins. Each item is characterized by two values, denoted by v_i^1 and v_i^2, where $i = 1, \ldots, n$. The problem is to find a minimum number of bins needed to pack all items, knowing that the capacity of each bin equals 1 in both requirements. We invite the reader to refer to (Spieksma, 1994) for more details.

2.3.6 Multi-dimensional bin packing problem

The three-dimensional bin packing problem is a generalization of 2D-BPP, arising when the depth of bins and items is taking into account. Only a very few studies have focused on 3D-BPP, among these we may cite the work of (Scheithauer, 1991), (Martello, et al., 2000), and (Gonçalves & Resende, 2013).

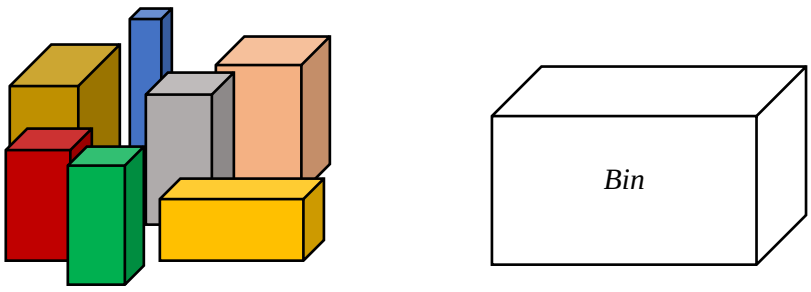


Figure 22. An example of the 3D-BPP instance

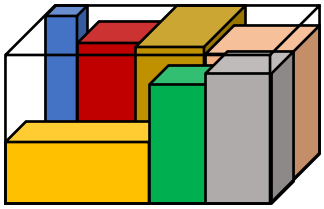


Figure 23. A possible solution of 3D-BPP

A much more generalized problem is the multi-dimensional (or *d*-dimensional) bin packing problem. (Amossen & Pisinger, 2010) have defined it as following: Given *d*-dimensional items and *d*-dimensional bins. Let V be a set of items and let $w : V \to \mathbb{R}_0^{+d}$ be a size function describing the width of each item in all dimensions. All bins are restricted to have the same size $W \in \mathbb{R}_0^{+d}$. A feasible solution of the problem consists of packing items into one or more bins so that no item exceeds the boundaries of the bin in which it is placed. The papers dealing with the multi-dimensional bin packing problem are extremely rare. For the sake of clarity, in the literature of the bin packing problem, the name "multi-dimensional bin packing problem" can also indicate the generalizing case of the vector bin packing problem.

3 Some related real-life situations

Applications of 2D-BPP and its variants are sundry, they can be found in industry, logistics and computer networks, just to mention few. A trivial possibility is to apply 2D-BPP in cutting problems, *e.g.* the items can be considered as the parts of clothes that have to be cut out of a piece of textile, while bins are regarded as the divided areas of the piece of textile. Such problems consist of determining an optimal cutting pattern so that the trim loss is minimized.

An interesting application is mentioned by (Stille, 2008) that is the delivery and vehicle routing problem wherein service devices move goods from service stations to delivery points. The service devices are vehicles, the delivery points are customers, the service stations are the company's storages, and the goods are the orders of customers. In the context of 2D-BPP, vehicles correspond to available bins and goods correspond to items that have to be packed. Each vehicle can only carry a certain volume and weight of goods, which represent the dimensions of each bin.

Likewise, the storage allocation problem arising in computer networks is also an important application presented in the work of (Stille, 2008). The storage system in computer networks is composed of several storage devices that can be characterized, for example, by access time, bandwidth, volume …, as well as several applications running independently on several computers. Each application requires a number of storage devices from the storage system to perform efficiently. Therefore, the storage allocation problem aims to find an optimal distribution of storage devices over applications. The problem can be modeled as a

multidimensional bin packing problem, where the storage devices are regarded as multidimensional bins and the applications are considered as the items.

Another application of 2D-BPP is illustrated by (Sarin & Wilhelm, 1984) in the context of layout design in robotic systems. In fact, to define layout configuration in robot systems, it must be resolved some facility design problems collectively. Among these problems, one aims to determine an optimal number of robots required to perform a set of tasks. This optimization problem has been modeled as 2D-BPP, where the tasks represent items and the robots correspond to available bins. Each robot has a limited capacity in space and a limited capacity to serve a certain demand, representing thus the two dimensions of available bins.

4 Lower bounds

4.1 The continuous lower bound

The lower bounds aim to assess the quality of solutions obtained by heuristic approaches. They can be also used as a stopping criterion for iterative algorithms and metaheuristics. Moreover, lower bounds can reduce the number of possible solutions in the search space. Many lower bounds have originally developed for 1D-BPP and then extended to 2D-BPP. We can cite, inter alia, the well-known continuous bound that can be calculated easily as below:

$$L_0 = \left\lceil \frac{\sum_{i=1}^{n} w_i \times h_i}{W \times H} \right\rceil \tag{5.24}$$

This lower bound yields likely good results for 1D-BPP, in particular, for small size items with respect to the bin size or when the size of items are uniformly distributed within $[0, H]$ (or [0, W]).

Let $Z(I)$ be the value of an optimal solution of an instance I of 2D-BPP, $L(I)$ the value provided by a proposed lower bound L, and let $r(I) = L(I)/Z(I)$. The absolute worst-case performance ratio of L is then defined as the smallest real number r such that $r(I) \geq r$ for any instance I of the problem. For 1D-BPP, the absolute worst-case performance ratio of the continuous lower bound is 1/2 according to (Martello & Toth, 1990). However, for 2D-BPP, the continuous lower bound has an absolute worst-case performance ratio equal to 1/4 according to (Martello & Vigo, 1998).

4.2 Lower bounds of Martello and Vigo

(Martello & Vigo, 1998) have proposed three lower bounds L_1, L_2, and L_3. The lower bound L_1 is a generalization case of their lower bound L_{1BP} proposed for the 1D-BPP. In fact, L_{1BP} decomposes the problem instances to small/ medium and large instances as following:

Let q be an integer such that $1 \leq q \leq \frac{1}{2} H$:

- $A_G = \{a_i \in A;\ H - q < h_i\}$
- $A_M = \left\{a_i \in A;\ \frac{1}{2}H < h_i \leq H - q\right\}$
- $A_P = \{a_i \in A;\ q \leq h_i \leq \frac{1}{2} H\}$

L_{1BP} can be computed by means of two lower bounds (L_α and L_β):

$$L_{1BP} = max\left(L_\alpha, L_\beta\right) \tag{5.25}$$

- The lower bound L_α is proposed by (Martello & Toth, 1990) and calculated by:

$$L_\alpha = \max_{1 \leq q \leq \frac{1}{2}H} \left\{|A_G \cup A_M| + max\left\{0, \left\lceil \frac{\sum_{a_i \in A_M \cup A_P} h_i}{H} \right\rceil - |A_M|\right\}\right\} \tag{5.26}$$

- The lower bound L_β is developed by (Dell'Amico & Martello, 1995) and calculated as following:

$$L_\beta = \max_{1 \leq q \leq \frac{1}{2}H} \left\{|A_G \cup A_M| + max\left\{0, \left\lceil \frac{|A_p| - \sum_{a_i \in A_M} \left\lfloor \frac{H - h_i}{q} \right\rfloor}{\left\lfloor \frac{H}{q} \right\rfloor} \right\rceil\right\}\right\} \tag{5.27}$$

An analogous lower bound L_1^W can be computed by considering the width of items and bins instead of their height. Notice that L_1^H is computed in the same manner as L_{1BP}. Finally, L_1 can be calculated as:

$$L_1 = max(L_1^W, L_1^H) \tag{5.28}$$

The lower bound L_2 explicitly considers both dimensions. Similarly to L_1, L_2 considers three sets of problem instances according to their width:

Let p be an integer such that $1 \leq p \leq \frac{1}{2} W$:

- $A_G = \{a_i \in A;\ W - p < w_i\}$
- $A_M = \left\{a_i \in A;\ \frac{1}{2}W < w_i \leq W - p\right\}$
- $A_P = \{a_i \in A;\ p \leq w_i \leq \frac{1}{2} W\}$

The lower bound L_2^W can be computed as below:

$$L_2^W = L_1^W + \max_{1 \leq p \leq \frac{1}{2}W} \left\{0; \left\lceil \frac{\sum_{a_i \in A_M \cup A_P} h_i w_i - \left(HL_1^H - \sum_{a_i \in A_G} h_i\right)W}{WH} \right\rceil\right\} \tag{5.29}$$

An analogous lower bound L_2^H can be produced by replacing the width of items and bins with their height. Hence, the lower bound L_2 can be calculated as:

$$L_2 = \max(L_2^W, L_2^H) \tag{5.30}$$

(Martello & Vigo, 1998) have shown that L_2 dominates both L_1 and L_0.

The lower bound L_3, similarly to L_2 and L_1, rests on three sets of problem instances which consider both dimensions as follows:

Let p and q be integers such that $1 \le p \le \frac{1}{2} W$ and $1 \le q \le \frac{1}{2} H$.

- $A_G = \{a_i \in A;\ W - p < w_i \text{ and } H - q < h_i\}$
- $A_M = \left\{a_i \in A \backslash A_G\ ;\ \frac{1}{2}W < w_i \text{ and } \frac{1}{2}H < h_i\right\}$
- $A_P = \{a_i \in A;\ p \le w_i \le \frac{1}{2} W \text{ and } q \le h_i \le \frac{1}{2} H\}$

The lower bound L_3 can be computed as below:

$$L_3 = \max_{\substack{1 \le p \le \frac{1}{2}W \\ 1 \le q \le \frac{1}{2}H}} \left\{ |A_G \cup A_M| + max\left\{0, \left\lceil \frac{|A_P| - \sum_{a_i \in A_M} m(a_i, p, q)}{\left\lfloor \frac{W}{p} \right\rfloor \left\lfloor \frac{H}{q} \right\rfloor} \right\rceil \right\} \right\} \tag{5.31}$$

Where,

$$m(a_i, p, q) = \left\lfloor \frac{H}{q} \right\rfloor \left\lfloor \frac{W - w_i}{p} \right\rfloor + \left\lfloor \frac{W}{p} \right\rfloor \left\lfloor \frac{H - h_i}{q} \right\rfloor - \left\lfloor \frac{H - h_i}{q} \right\rfloor \left\lfloor \frac{W - w_i}{p} \right\rfloor \tag{5.32}$$

(Martello & Vigo, 1998) have proved that no dominance relation exists between L_3 and L_2.

4.3 Lower bound of Fekete and Schepers

(Fekete & Schepers, 2000) have proposed a new lower bound that dominates the ones proposed by (Martello & Vigo, 1998) for 2D-BPP. They are based on the concept of Dual Feasible Function (DFF) [23] that consists of transforming the items size in such a way that any transformed instance can still be packed if the original instance could be packed. Then, the lower bound is obtained by calculating the continuous lower bound L_0 using the new areas of the resulting items.

5 Exact algorithms

The 2D-BPP is NP-hard in the strong sense since it is a generalization of the well-known one-dimensional bin packing problem, which is according to (Johnson & Garey, 1979) is an NP-hard optimization problem. Consequently, exact methods cannot calculate optimal

[23] *A function* $u : [0, 1] \to [0, 1]$ *is called dual feasible function if, for any finite set S of nonnegative real numbers, we have the relation:* $\sum_{x \in S} x \le 1 \Rightarrow \sum_{x \in S} u(x) \le 1$.

solutions in reasonable runtime for such problems, especially for large-scale instances. Only a few exact approaches for 2D-BPP are known so far and their applicability is limited to sets with up to 120 items. For example, (Martello & Vigo, 1998) have embedded new lower bounds within an enumerative algorithm to find exact solutions. The experimental results showed that the exact algorithm could resolve efficiently the problem instances containing up to 120 items. (Clautiaux, et al., 2007) have proposed two exact algorithms: The first algorithm is an improved version of the branch and bound method, which is able to handle redundancies occurring in the classical branch and bound method. However, the second algorithm is based on a new relaxation of the problem that can detect non-feasible instances. By this way, this latter algorithm can reduce the number of visited nodes in comparison to that proposed by (Martello & Vigo, 1998). The computational experiments were carried out on randomly generated instances containing up to 23 items, indicating the efficiency of both proposed methods. Furthermore, (Pisinger & Sigurd, 2007) have developed a hybrid approach that combines branch-and-price with constraint programming algorithm. They have applied the column generation principle and solved the specific pricing problem by means of constraint programming. Their proposed approach was tested on the benchmark instances proposed by (Berkey & Wang, 1987) and (Martello & Vigo, 1998).

6 Heuristics

In contrast to exact methods, heuristics have been deemed most suitable for solving 2D-BPP. Most of them are based on the classical heuristics of 1D-BPP, including:

- Next Fit (NF) heuristic: This heuristic places iteratively each item in the lowest position of the first bin that can accommodate it. If an item cannot be packed in the current bin, a new bin is created and the previous bin is removed from the list of available bins. Next Fit Decreasing (NFD) heuristic is a special case of NF heuristic in which the items to be packed are initially sorted in decreasing order of their height.

- First Fit (FF) heuristic: It uses the same procedure as in NF heuristic, except that if an item cannot be packed in the current bin, a new bin is created and the previous bin is retained to receive the remaining items. First Fit Decreasing (FFD) heuristic is a special case of FF heuristic in which the items to be packed are initially sorted in decreasing order of their height.

- Best Fit (BF) heuristic: It consists of placing items in bins having the lowest gap (*i.e.* non-occupied areas) that can accommodate them. Best Fit Decreasing (BFD) heuristic is a special case of BF heuristic in which the items to be packed are initially sorted in decreasing order of their height.

According to (Lodi, et al., 2002), heuristics dedicated to 2D-BPP can be classified in two families: One-phase algorithms in which items are directly packed into bins, and two-phase

algorithms that start by packing items in an open-ended strip and then the obtained strip solution is used to construct a packing into finite bins. In other words, the first phase consists of solving a 2D strip packing problem, while the second phase involves solving the 1D-BPP.

6.1 One phase algorithms

The one-phase heuristics consist of packing items iteratively in bins by taking into account two main rules: The order in which the items are treated and the place wherein the items will be packed in bins. Among one-phase heuristics, those that pack items from left to right in levels, called level-oriented heuristics. Some examples of this latter were presented and experimentally evaluated by (Berkey & Wang, 1987): The Finite First Fit heuristic (FFF) which consists of sorting items in decreasing order of their height. The first item initializes the first height level in the first bin. Thereafter, each item is added to the lowest level of the first bin in which it fits. If it cannot be packed in any level of the current bin, a new bin is initialized with a new height level. Notice that, all created bins are retained in the list of available bins (see Figure 24). Otherwise, Finite Next Fit heuristic (FNF) rests on the same principle of FFF heuristic, except that if no level in the first bin can accommodate a given item, a new bin is initialized with a new height level, while the previous bin is removed from the list of available bins (see Figure 25).

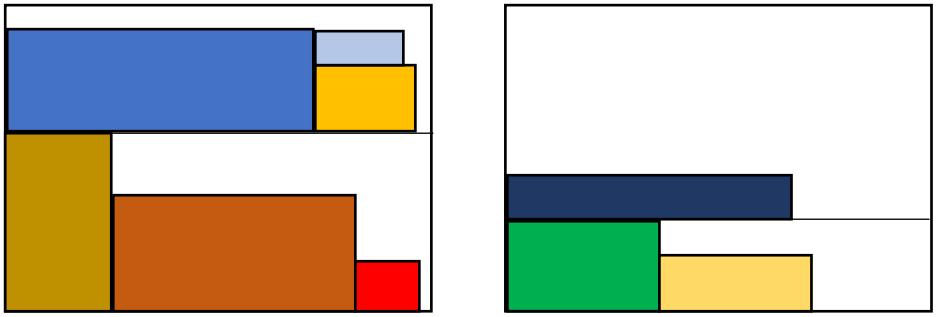


Figure 24. FFF heuristic

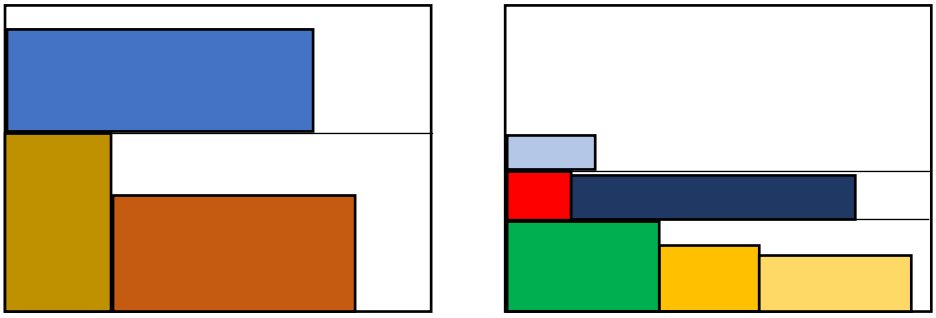


Figure 25. FNF heuristic

However, Finite Bottom-Left strategy developed by (Berkey & Wang, 1987), is an example of heuristics that are not based on packing by levels. Indeed, the items are considered in a given order and each item is placed in the lowest and the left-most position that accommodates it. If there is no bin in which the item can be packed, a new bin is created (see Figure 26). Another

example is the Next Bottom-Left strategy proposed by (Berkey & Wang, 1987) which is similar to Finite Bottom Left strategy, except that if the current bin cannot accommodate a given item, a new bin is initialized and the previous bin is removed from the list of available bins (see Figure 27). In addition to Alternate Directions (AD) heuristic that is developed by (Lodi, et al., 1999). It sorts firstly the items in decreasing heights and computes a lower bound *L*. Then, *L* bins are initialized by packing on their bottom border a subset of items according to BF strategy. The remaining items are packed into non-horizontal bands alternatively from left to right and from right to left.

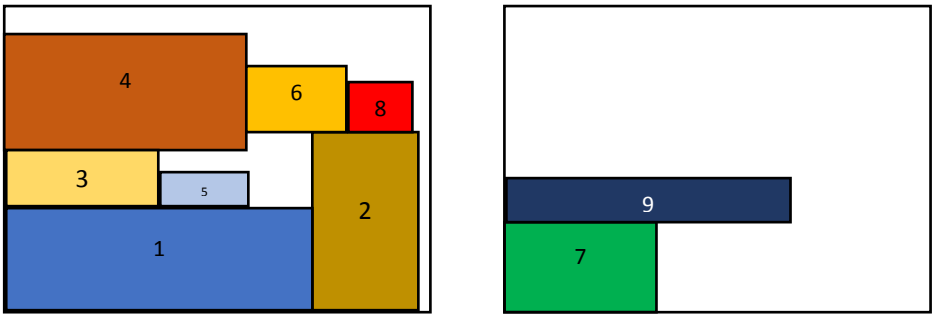


Figure 26. Finite Bottom Left heuristic

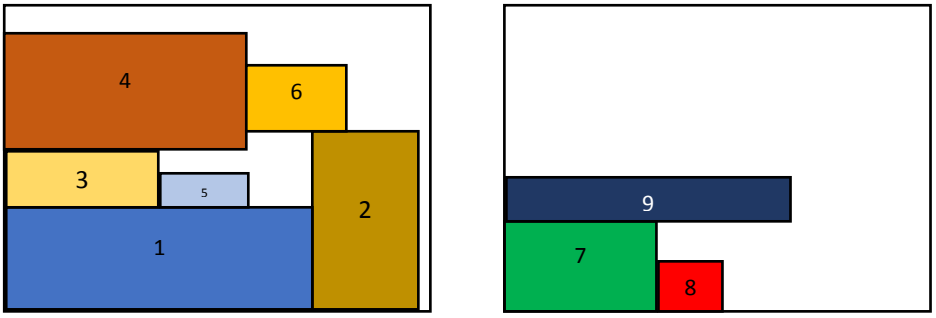


Figure 27. Next Bottom Left heuristic

6.2 Two-phase algorithms

The majority of two-phase heuristics are based on the following strategies: Next Fit Decreasing Height (NFDH), First Fit Decreasing Height (FFDH), and Best Fit Decreasing Height (BFDH). These strategies have been derived from the famous NF, FF, and BF strategies of 1D-BPP. Initially, the items are sorted by decreasing heights. Then, the NFDH, FFDH, or BFDH heuristics pack items by level in an open-ended strip by using respectively NF, FF, or BF strategies. The first level is the bottom of the strip and subsequent levels are defined by the tallest item packed at the level below. The items are placed in the left as possible and the lowest level that can accommodate them (see Figure 28).

A two-phase heuristic has been proposed by (Chung, et al., 1982), called Hybrid First-Fit (HFF) heuristic. In the first phase, the HFF heuristic makes use of the FFDH heuristic to pack items by level into an open-ended strip, thereby obtaining a collection of blocks of decreasing heights.

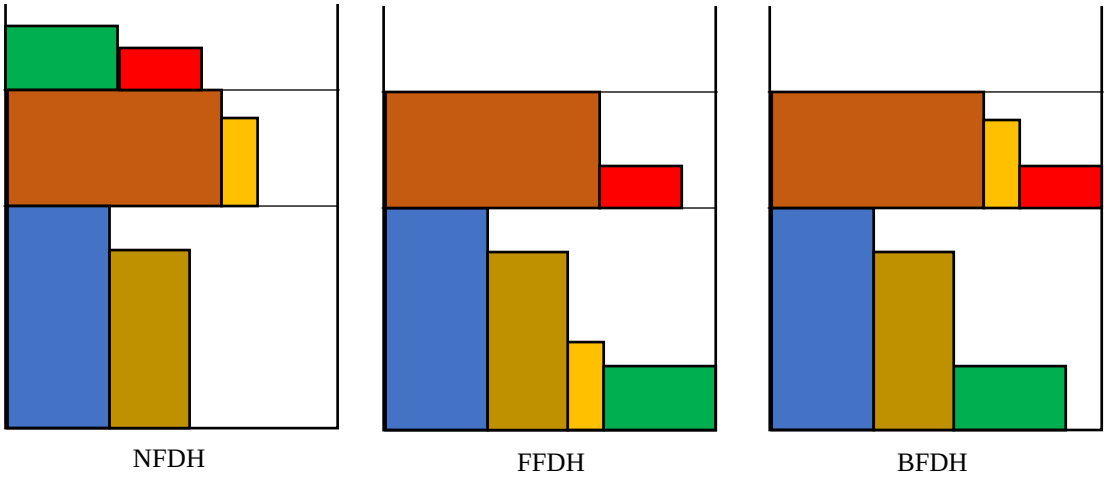


Figure 28. NFDH, FFDH, and BFDH heuristics

The width of the block is the width of the strip, while the height of the block is the height of the tallest item placed on that level. In the second phase, the FFD heuristic is applied to pack the resulting blocks into finite bins by considering them as an instance of 1D-BPP. (Berkey & Wang, 1987) have proposed a variant of HFF heuristic, called Finite Best Strip (FBS), in which they apply BFDH heuristic to pack items in the first phase and the BFD heuristic to pack blocks in the second phase. Similarly, (Frenk & Galambos, 1987) have proposed a variant of HFF heuristic, called Hybrid Next Fit heuristic, which uses the same idea in conjunction with NFDH heuristic in the first phase and the NFD heuristic in the second phase.

On the other hand, (Lodi, et al., 1999) have proposed different heuristics, including the Floor-Ceiling (FC) and the Knapsack Packing (KP) heuristics. In the first phase, the FC heuristic consists of packing items from left to right with their bottom edge on the level floor (*i.e.* the bottom horizontal line), or from right to left with their top edge touching the level ceiling (*i.e.* the horizontal line drawn on the top of the tallest item packed on the floor). In the second phase, the resulting strip levels are packed into finite bins using an exact method for solving 1D-BPP. Otherwise, in the first phase of the KP heuristic, they consider one level at a time. A level is initialized by the tallest item, and the remaining items are packed by solving an instance of the one-dimensional knapsack problem that aims to pack items as maximum as possible in each strip level. Each item is characterized by a weight corresponding to its width and a profit corresponding to its surface, while the capacity of the knapsack is the difference between the width of the strip and the width of the tallest item. In the second phase, similarly to FC heuristic, an exact algorithm is used for solving the 1D-BPP instance. These described heuristics are generally applicable to several variants of 2D-BPP.

A relatively recent heuristic has been developed by (Monaci & Toth, 2006), which also operates in two phases. In the first phase (or column generation phase), it generates a very large number of feasible subsets of items (Columns) that can be packed into a single bin by means of greedy heuristics. In the second phase (or column-optimization phase), a feasible solution of the

problem is obtained by solving the associated set-covering problem[24] through Lagrangian relaxation heuristic.

7 Metaheuristics

To enhance the search ability hoping to find high-quality solutions, recent studies are widely focalized on metaheuristic approaches. For instance, (Lodi, et al., 1999) have developed a unified Tabu Search (TS) approach that uses different heuristics separately (AD, KP…), in order to evaluate the moves in the neighborhood search phase. Their proposed TS is tested on the test instances proposed by (Berkey & Wang, 1987) and (Martello & Vigo, 1998), indicating that the results found are satisfactory. (Puchinger & Raidl, 2004) have proposed a branch and price (B&P) approach that incorporates an evolutionary algorithm for solving the pricing problem. The performance of their approach was investigated through the benchmark instances of (Berkey & Wang, 1987) and (Martello & Vigo, 1998). The outcome results showed that the combination of B&P and an evolutionary algorithm is a good alternative to solve 2D-BPP to optimality. In addition, in the work of (Parreño, et al., 2010), a new Greedy Randomized Adaptive Search Procedure (GRASP) is designed. The constructive phase is based on the maximal-space concept[25], while in the improvement phase, several new moves are combined with Variable Neighborhood Descent (VND). The simulations demonstrated that the proposed hybrid approach is equal or better than the best state-of-art heuristics, over the benchmark instances of 2D-BPP as well as 3D-BPP. In the same way, (Gonçalves & Resende, 2013) have suggested a new random-key genetic algorithm[26] (RKGA) for 2D-BPP and 3D-BPP, which is based on the maximal-space concept to manage free space in bins. The experiments tested on 858 benchmark instances proved that the resulting algorithm performs equally or better than other approaches published in the literature. (Soke & Bingul, 2006) have combined an improved Bottom Left heuristic with a genetic algorithm (GA) and a simulated annealing (SA) method. The empirical experiments have shown that the hybrid GA performs better than the hybrid SA for solving 2D-BPP. Besides, (Blum & Schmid, 2013) have developed an evolutionary algorithm that incorporates a heuristic proposed by (Wong, 2009). Experiments were carried out on 500 instances containing 20 to 100 items characterized by various sizes and bins capacities. Thus, the results obtained showed that the proposed algorithm is competitive in comparison with some previously existing approaches. On the other side, only a few researchers have been interested in swarm intelligence algorithms to solve 2D-BPP. A particle swarm optimization (PSO) algorithm dealing with multi-objective 2D-BPP is introduced by (Liu, et al., 2006) and (Liu, et al., 2008). More recently, an improved version of PSO algorithm is suggested by (Shin & Kita, 2017). It uses in addition to the global and the personal best

[24] The set covering problem consists of partitioning a given set of items into subsets having peculiar characteristics so that the sum of subsets costs is minimized.

[25] The empty area is maximal if it is not contained in any other area in the bin.

[26] RKGA is a genetic algorithm in which the chromosomes are represented as vectors of randomly generated real numbers in the interval [0,1].

positions, a second global best position of all particles that is determined in a probabilistic way. The simulations indicated that the improved PSO could increase the number of packed items although the running time is increased.

8 Conclusion

This chapter briefly introduced the two-dimensional bin packing problem and discussed its mathematical formulations. Then, it provided background about the extensions and variants of 2D-BPP, and highlighted some of its real-life applications. In addition, the current chapter presented some important lower bounds and discussed the existing exact methods and heuristic approaches devoted to resolving 2D-BPP.

Chapter 6: A binary crow search algorithm for solving 2D-BPP

1 Introduction

Although various metaheuristics have been designed for solving 2D-BPP, especially for large scaled-instances, only a few articles apply the so-called swarm intelligence algorithms to deal with 2D-BPP despite their success and simplicity. Indeed, swarm intelligence algorithms consider nature as an unending source of inspiration. The behavior of bees, virus, glowworms, fireflies, and other organisms have inspired swarm intelligence researchers to develop new optimization algorithms. A large and growing body of swarm intelligence algorithms are devoted to solving continuous optimization problems like the algorithm of (Pham, et al., 2005), (Askarzadeh, 2016), and (Liang & Cuevas Juarez, 2016), because continuous optimization problems tend to be easier to solve than discrete optimization problems.

However, a much more of the current literature of swarm intelligence algorithms are adapted to binary search space by means of binarization techniques (Crawford, et al., 2017) like Transfer function, set-based approach, and quantum approach…. We might cite as examples of binarized algorithms the particle swarm optimization algorithm (Kennedy & Eberhart, 1997), glowworm search algorithm (Mingwei, et al., 2014), and fruit fly optimization algorithm (Wang, et al., 2013). Such algorithms can be competitive optimization tools for tackling with combinatorial problems.

In this chapter, we choose for solving 2D-BPP, a recent algorithm introduced by (Askarzadeh, 2016) that is based on swarm intelligence principles and called Crow Search Algorithm (CSA). The CSA algorithm successfully applied to continuous optimization problems and paid off in terms of solution quality, mostly in convergence speed since it is a less parameter algorithm. In fact, the CSA algorithm simulates the intelligence of crows in hiding excess food for a future need. Hiding food source by a crow is not an easy task because crows are known for their thievery behavior. In other words, the crows follow each other to steal the hidden foods of their colleagues.

The structure of this chapter is organized as following. Section 2 provides some examples of swarm intelligence algorithms. Section 3 presents a background of particle swarm optimization algorithm (that is regarded as a comparative method in this chapter) and surveys the studies that have been interested in this latter. In section 4, we start by providing the principles of the CSA algorithm, and then we give an overview of some related work. The next section describes in detail the proposed binary version of CSA algorithm for solving 2D-BPP. An evaluation of the binary CSA is carried out on benchmark instances and the experimental results are reported in section 6. Finally, we conclude with some recommendations for future work in section 7.

2 Swarm intelligence algorithms

The swarm intelligence (SI) algorithms belong to the class of bio-inspired algorithms that fall under the large class of nature-inspired algorithms. The nature-inspired algorithms inspire

from the complex and highly organized elements of nature, and simulate the cooperative behavior and sometimes the interdependence of natural elements such as plants, animals, insects, and so on. However, the bio-inspired algorithms imitate the biological phenomena. A taxonomy based on the inspiration source of algorithms (Goel, et al., 2014) is illustrated in Figure 29.

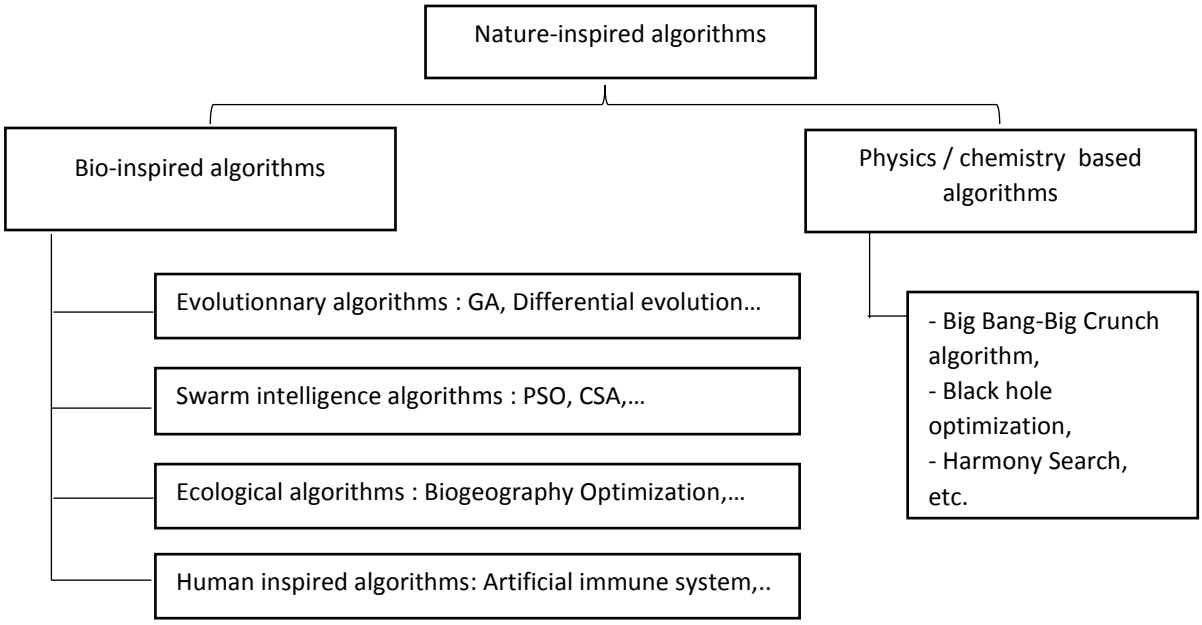


Figure 29. A taxonomy of nature-inspired algorithms

The SI algorithms emulate the real-world behavior of swarms like birds, fishes, termites, etc. They have multiple agents that work together following a self-organized process and interacting in a cooperative manner. SI algorithms are popular and widely used in optimization problems. The early algorithms based on swarm intelligence's aspect were ant colony optimization algorithm (Dorigo, et al., 1996) and particle swarm optimization algorithm (Kennedy & Eberhart, 1995). Then, enormous SI algorithms have appeared inspired by fish schools (Cai, 2010), as well as different aspects of the behavior of insects (Feng, et al., 2009), plants (Uymaz, et al., 2015), and animals (Yang, 2010), etc. Despite the swarm inspiration common to these algorithms, each one has its own process to exploit and explore the search space of a given problem. Most of SI algorithms were devoted to tackling with continuous optimization problems, while some of them were later adapted to be applied to discrete optimization problems in general and binary problems in particular. Let us briefly discuss some examples of algorithms based on swarm intelligence principles.

Bat algorithm. It was introduced by (Yang, 2010) and applied to a benchmark of mathematical functions. The algorithm is inspired by the echolocation behavior of bats. The bats use a type of sonar (called echolocation) in order to sense distance, detect prey, and avoid obstacles in the dark. A swarm of bats (possible solutions) fly randomly through a search space hoping to find food or prey (best solution) while using echolocation to update their positions and velocities.

Mosquito host-seeking algorithm. It was introduced by (Feng, et al., 2009) for solving the traveling salesman problem. The algorithm is inspired by the host-seeking behavior of mosquitoes and based on three principles: The mosquito looks for carbon dioxide or smelling substance; It distinguishes the smell it loves, and then seeks towards a high-concentration location; It makes a descent when it feels the radiant heat[27] of the host. A swarm of mosquitoes (possible solutions) is distributed surrounding a host. Each mosquito is being attracted to seek towards the host by carbon dioxide, smelling substance, and radiated heat. When all mosquitoes stop moving, being in an equilibrium state, the algorithm achieves the global optimum.

Artificial Fish School Algorithm. It was developed by (Cai, 2010) for solving the berth allocation problem. The algorithm is inspired by the foraging process of fish. It is based on three basic principles: The fish perceive the concentration of food in water to determine the movement by vision or sense and then chooses the tendency; The fish swim in groups in order to protect themselves from their predators; In the moving process of the fish swarm, when a single fish or several fish find food, the neighborhood fish will trail and reach the food quickly. The fish correspond to possible solutions of the problem at hand, while the place at which there are lots of food represents the best solution.

Artificial Algae algorithm. It was introduced by (Uymaz, et al., 2015) and applied to a benchmark of mathematical functions and a constrained design optimization problem. This algorithm simulates algae reproduction, adaptation to the environment, and their movements for being close to the light as a photosynthetic organism[28]. The algae correspond to the possible solutions of the problem at hand, while the point on which algae can receive optimum light for photosynthesis represents the global optimum of this algorithm.

Other SI algorithms (*e.g.* the crow search algorithm and the particle swarm optimization algorithm) are described in detail in the next sections.

3 The particle swarm optimization algorithm

Particle swarm optimization (PSO) algorithm is a SI based approach inspired by the social behavior of bird flocking or fish schooling. It has been developed by (Kennedy & Eberhart, 1995) for solving continuous optimization problems. The PSO algorithm has been successfully applied to solve several optimization problems like scheduling, traveling salesman, network deployment, task assignment, and neural network training problems.

3.1 The principle of PSO algorithm

The PSO algorithm considers the particles as potential solutions of the optimization problem at hand. Each particle i in the swarm has a position and a velocity in a D-dimensional

[27] The heat energy transmitted by electromagnetic waves.

[28] The organism that uses sunlight to synthesize nutrients from carbon dioxide and water.

space, denoted respectively $p_i(t)$ and $v_i(t)$ ($i \in \{1, \dots, N\}$) where N is the number of particles in the swarm and t is the iteration time. The element of the position vector $p_{id}(t)$ ($d \in \{1, \dots, D\}$) corresponds to decision variables of the problem. The satisfaction of each particle's position is measured by a fitness function $f(p_i(t))$. The best position of each particle through iterations is known as the personal best position $p_i^{pbest}(t)$, while the best position ever found by the whole swarm is known as the global best position $p^{gbest}(t)$. The position of particles and their velocity are updated by following rules:

$$p_i(t+1) = p_i(t) - v_i(t+1) \tag{6.1}$$

$$v_i(t+1) = \omega . v_i(t) + c_1 r_1 \left(p_i^{pbest}(t) - p_i(t) \right) + c_2 r_2 \left(p^{gbest}(t) - p_i(t) \right) \tag{6.2}$$

Where ω is the inertia weight, c_1 and c_2 are the acceleration coefficients, while r_1 and r_2 are two randomly generated numbers in the interval [0,1]. Equation (6.1) indicates how a particle updates its position by means of the velocity vector. Equation (6.2) represents the update rule of the velocity vector that contains three components: Inertia component that makes the particle moves at the same direction with the same velocity as the neighbor particles, in addition to personal influence component that improves the quality of each particle's position by giving the possibility to each particle to return to the previous position if it is better than the current one, as well as a social influence component that makes particles follow the best neighbors direction. It should be emphasized that the inertia weight is generally not fixed, it is varied as the algorithm progresses by the given formula:

$$\omega = \omega_{max} - (\omega_{max} - \omega_{min}) \times \frac{t}{t_{max}} \tag{6.3}$$

Where ω_{max} and ω_{min} denote respectively, the maximum and minimum inertia weight, while t and t_{max} are respectively, the iteration time and the maximum of iterations. The pseudo code of PSO algorithm is presented in Algorithm 7.

1. **Input*: N* particles, the PSO parameters (ω, c_1, c_2,…)**
2. Initialize positions of particles with random numbers
3. Evaluate the position of particles
4. Initialize the personal best position of each particle
5. Initialize the global best position in the swarm
6. Initialize velocities of particles with random numbers between [Vmin, Vmax] (where Vmax and Vmin are respectively, predetermined maximum and minimum velocity, generally we preset Vmin= - Vmax)
7. Set $t = 1$
8. Repeat
9. For $i = 1\ to\ N$
10. Evaluate the fitness function of each particle
11. If $f(p_i(t+1)) > f(p_i^{pbest}(t))$ then $p_i^{pbest}(t+1) = p_i(t+1)$
12. Find the best position p^1 in the whole swarm

13. If $f(p^1(\mathrm{t}+1)) > f(p^{gbest}(t))$ then $p^{gbest}(t+1) = p^1(\mathrm{t}+1)$
14. Update the velocity of each particle
15. Update the position of each particle
16. End For
17. Set $t = t + 1$
18. Until $t \geq tmax$
19. **Output: A global best position**

Algorithm 7. The pseudo code of PSO algorithm

3.2 The binary PSO algorithm

The first binary version of PSO, called Binary Particle Swarm Optimization (BPSO) algorithm, was introduced by (Kennedy & Eberhart, 1997) in order to handle binary optimization problems. In the binary version, the positions and velocities are defined in terms of changes of probabilities. Each element of the velocity vector represents a rate of change of its corresponding element in position vector, while the elements of the position vector are restricted to value 1 or 0. In other words, an element of the velocity vector indicates the probability P that the corresponding position element takes the value one, and $1 - P$ means that it takes the value zero. To transform the velocity vector to a probability vector, (Kennedy & Eberhart, 1997) have used a Sigmoid function given below:

$$sig(x) = \frac{1}{1+e^{-x}} \quad ; \forall\, x \in \mathbb{R} \tag{6.4}$$

The resulting change is defined as following:

$$p_{id}(t) = \begin{cases} 1 & if \;\; sig(v_{id}(t)) \geq rand\,() \\ 0 & otherwise \end{cases} \qquad \forall\, i \in \{1, \dots N\}\; and\; d \in \{1, \dots, D\} \tag{6.5}$$

Where $rand()$ is a random number generated in the interval [0,1].

To investigate the performance of BPSO algorithm, the authors have used a suite of benchmark functions considered to be reputable and thorough. Thus, the outcome results demonstrated that the proposed binary version of PSO algorithm is characterized by a fast convergence speed.

Later, several variants of binary PSO algorithm have been developed to work in binary search space of solutions like the Quantum-inspired BPSO of (Jeong, et al., 2010), Chaotic binary PSO of (Wang, et al., 2012)), etc. However, we invite the reader to refer to the article of (Crawford, et al., 2017), in which they have surveyed an exhaustive list of existing binarization techniques designed for continuous problems by providing in detail their definitions.

4 The crow search algorithm

The crow search algorithm (CSA) is a novel SI based metaheuristic introduced by (Askarzadeh, 2016), and inspired by the social intelligence of crows and their food gathering

process. The CSA is formulated on the following assumptions: (a) Crows live in groups; (b) Crows are very greedy birds and strategists, they hide food and are able to recover them until months later; (c) Crows are known to be thieves, they follow each other for stealing food and; (d) Crows are cautious, they make use of their experiences as a thief in predicting pilferer's behavior for protecting their own food source.

4.1 Related work

The CSA is shown to be very effective in solving complex optimization problems in terms of quality of solutions and convergence speed. Furthermore, the experiments proved its high performance against other popular algorithms such as GA, PSO algorithm, and artificial bee colony algorithm. This subsection presents some works that have been interested in CSA for solving different kinds of optimization problems.

(Sheta, 2017) has used CSA to solve the economic load dispatch (ELD) problem that occurs in power system planning. The empirical experiments were carried out on three power generators test cases, indicating that the CSA performs better than GA, PSO algorithm, and flower pollination algorithm of (Abdelaziz, et al., 2016). Also, (Mohammadi & Abdi, 2018) have been interested in solving ELD problem. In this regard, they have proposed a modified CSA (MCSA) in which they have used a selection method that determines how each crow should choose another crow from the swarm to adjust its direction in the search space. Besides, they have incorporated an intelligent method to control the flight length of crows (a main parameter in CSA). The MCSA was tested on 6, 10, 40, 110, and 160 unit test systems, then it was compared with other existing algorithms in terms of solution quality and computing time. The results demonstrated the superiority of MCSA against other comparative methods.

(Adhi, et al., 2018) have applied CSA to solve scheduling problems in an acceptable runtime. The simulation results indicated that the CSA yields a good quality of solutions in reasonable runtime against other algorithms designed for the same purpose including two variants of GA and a PSO algorithm, as well as a parallel version of Variable Neighborhood Search algorithm.

(Díaz, et al., 2018) have suggested an improved version of CSA to handle complex optimization problems of energy in electrical systems. Their proposed algorithm aims to improve the convergence of high multi-modal formulations while preserving the diversity of original CSA. To investigate the performance of the improved CSA, it was tested on two complex high multi-modal optimization problems that are the identification of induction motors and capacitor allocation in distribution networks. The obtained results demonstrated the best performance of their algorithm in comparison to three comparative methods.

(Jain, et al., 2017) have developed another improved version of CSA for high-dimensional global optimization problems. They have incorporated the Lévy flight behavior of birds in CSA in order to improve the ability of the global search in solutions space. Also, they have used an

experience factor and adaptive adjustment operator to update the positions of crows. The experiments tested on 15 benchmark functions containing up to 500 dimensions, indicated that their proposed algorithm performs better than CSA as well as other comparative methods including GA and PSO algorithm in terms of accuracy and convergence speed.

On the other side, a few authors have been interested in adapting CSA to binary optimization problems. Among them, we cite (De Souza, et al., 2018) that have proposed a binary version of CSA based on "V-shaped" binarization technique ((Mirjalili & Lewis, 2013), (Crawford, et al., 2017)). They have applied their proposed algorithm to tackle with an optimization problem of feature selection that consists of identifying and separating relevant features (excluding redundant and noisy features) in order to classify data into a minimum number of clusters. To test the performance of their binary CSA, it was applied to six benchmark data sets and compared with some classical state-of-art algorithms including two swarm intelligent algorithms and two greedy algorithms. The results showed that the proposed algorithm achieves very good results in terms of classification accuracy and the number of selected features while consuming relatively low runtime.

4.2 The principle of CSA algorithm

It is assumed that there is a d-dimensional environment including N crows. The position of each crow i at time t in the search space having d dimensions, is represented by a vector $x_i(t) = (x_i^1(t),\ x_i^2(t), \dots, x_i^d(t))$, where $i \in \{1, \dots, N\}$ and each x_i indicates a feasible solution of the optimization problem. In addition to positions, each crow i has a memory $m_i(t)$ at time t, and can hence remember the best position in which it has stored its food. The feasibility of the position of each crow is evaluated by the fitness function of the optimization problem at hand.

The CSA starts with randomly generated positions of crows and sets the memory of crows to their initial positions. Then, resting on the thievery behavior that crows exhibit, each crow updates its position as following: The crow i randomly selects a crow j from the swarm and follows it to discover where the latter has stored its food. If the crow j perceives the presence of crow i, the former fools the latter by moving towards a random position. The new position of the crow i is obtained by the following equation:

$$x_i(t) = \begin{cases} x_i(t-1) + r_i \times fl_i \times (m_j(t-1) - x_i(t-1)) & if\ r_j \geq AP_j \\ a\ random\ position & otherwise \end{cases} \quad (6.6)$$

Where fl_i is the flight length of the crow i, and AP_j is the probability of awareness of the crow j. Besides, r_i and r_j are uniform distributed random numbers in the interval [0,1]. The awareness probability of crows controls the diversification and intensification of CSA. Small values of AP_j lead to a search in local regions, thereby increasing intensification. On the other hand, the large values of AP_j lead to explore more the search space, thereby increasing the diversification.

Notice that the parameters AP_j and fl_i take constant values for all i and j, that's why some authors denoted these two parameters by AP and fl.

Thereafter, the algorithm checks the feasibility of the crows' positions (if the new position is infeasible, the crow stays in its current position). Then, according to the fitness function value, each crow updates its memory as described below:

$$m_i(t) = \begin{cases} x_i(t) & if\ f(x_i(t)) \geq f(m_i(t-1)) \\ m_i(t-1) & otherwise \end{cases} \quad (6.7)$$

Where $f(.)$ is the fitness value.

The CSA is stopped when a maximum number of iterations (t_{max}) is reached. In this step, the best memory in terms of the fitness value is reported as the final solution of the optimization problem. The pseudo-code of the original CSA algorithm is given in Algorithm 8.

1. **Input*: N* crows, the CSA parameters ($AP, fl, and\ tmax$)**
2. Randomly initialize the positions of crows
3. Evaluate the positions of crows
4. Initialize the memory of each crow
5. Set $t = 1$
6. Repeat
7. For $i = 1\ to\ N$
8. Randomly choose a crow j from the swarm to follow it
9. Define the awareness probability
10. If $r_j \geq AP_j$ then $x_i(t+1) = x_i(t) + r_i \times fl_i \times (m_j(t) - x_i(t))$
11. Else $x_i(t+1) = a\ random\ position\ from\ the\ search\ space$
12. End for
13. Check the feasibility of new positions
14. Evaluate the position of each crow
15. Update the memory of each crow
16. Set $t = t + 1$
17. Until t ≥ tmax
18. **Output: The best memory for the whole swarm**

Algorithm 8. The pseudo code of CSA algorithm

5 A binary version of the crow search algorithm

In original CSA, crows move in a continuous *d*-dimensional environment in which each variable takes a value within a continuous real domain. As the CSA has proved its accuracy and efficiency when it was applied to continuous optimization problems, we have extended it to binary optimization problems, in particular the 2D-BPP. The following subsections will elaborate on the sequential mechanisms of the proposed binary version of CSA (BCSA), while the main steps of BCSA are illustrated in Figure 30.

5.1 Initialization of crows' positions

At the initialization stage, N crows are randomly positioned in d-dimensional search space. The crow position (solution) is presented by a binary matrix of d columns and n rows, where n is the number of items to be packed into bins, and d corresponds to the number of available bins. The representation of each solution i at time t is defined as follows:

$$X^i(t) = \left\{x_{jk}^i(t)\ ;\ 1 \leq\ j \leq\ n\ and\ 1 \leq\ k \leq\ d\ \right\};\qquad For\ i \in \{1, \dots, N\} \tag{6.8}$$

Where

$$x_{jk}^i(t)\ =\ \begin{cases} 1 & if\ the\ item\ j\ is\ packed\ in\ the\ bin\ k \\ 0 & otherwise \end{cases};\qquad For\ i \in \{1, \dots, N\} \tag{6.9}$$

Notice that the memory M^i of each crow i is initialized by its initial position since it has as yet no experience at this stage.

5.2 Checking the feasibility of solutions

Since the positions of crows are randomly generated, some infeasible solutions may occur in the search space. A solution is considered as infeasible when it violates the capacity constraints of used bins. Therefore, for dealing with infeasible solutions, a problem specific-knowledge repair operator is developed. This repair operator is applied only to overloaded bins. It consists firstly of removing items from overloaded bins in turn so as to respect their loading capacities. Then, it inserts the removed items one by one in the former well-filled bins that can accommodate them by using the Bottom-Left strategy.

5.3 Binarization process

In BCSA, we limit the positions of crows only to binary values by using a technique called "the Sigmoid function" that converts the real-valued positions into binary ones. In other words, the Sigmoid function defines the probability of changing the elements of a position vector from 0 to 1 and vice versa.

By applying the Sigmoid function, the resulting change in each crow's position is defined by the following rules:

I. If $r_j \geq AP_j$

i. Calculate the following function :

$$g^i(t-1) = r_i \times fl_i \times \left(M^j(t-1) - X^i(t-1)\right);\quad \forall\ i \in \{1, \dots, N\} \tag{6.10}$$

ii. Calculate the Sigmoid function :

$$sig(g_{jk}^{i}(t-1)) = \frac{1}{1+e^{(-g_{jk}^{i}(t-1))}} ; \forall\, i \in \{1,\dots,N\};\, j \in \{1,\dots,n\};\, k \in \{1,\dots,d\} \qquad (6.11)$$

iii. Then, apply equation (6.11) to discretize X^i at time t for each element x_{jk}^{i}.

II. Else

For each crow i, its corresponding position is randomly updated according to the following rule:

$$\forall\, 1 \le k \le d \; and \; 1 \le j \le n \; ; \; x_{jk}^{i}(t) = \begin{cases} 1 & if \quad rand() \ge 0.5 \\ 0 & sinon \end{cases} \qquad (6.12)$$

Then, we check the feasibility of the new generated positions. If it is necessary, we apply the proposed repair operator to deal with infeasible solutions.

5.4 Evaluation of solutions

In most cases, the fitness function of the bin packing problems differs from the objective function. In fact, minimizing the number of bins used to pack all items is correct from a mathematical point of view but it is unusable in practice. Thus, to evaluate and assess the quality of solutions, the fitness function introduced by (Falkenauer & Delchambre, 1992) is used by adapting it to the context of 2D-BPP:

$$f = \frac{\sum_{u=1}^{usedBin}(fill_u/C)^z}{usedBin} \qquad (6.13)$$

Where $usedBins$ is the number of bins used to pack all items, $fill_u$ is the total amount of surface area covered by the items placed in bin u. C is the capacity of each bin in terms of surface, and z is a constant (> 1) which indicates the concentration in the well-filled bin compared to the less-filled bin. The experimental results carried out by authors have shown that $z = 2$ yields good results. This fitness function has to be maximized. It is based on the fact that fulfilling bins as maximum as possible with available items may reduce the number of used bins.

5.5 Updating of crows' memory

Once the quality of solutions is evaluated by the equation (6.13), the memory of each crow at time t is updated with equation (6.7). If the new position of a crow is better than the most recently memorized position, the crow updates its memory with this new position.

5.6 Termination criterion

The previous steps are repeated until a maximum number of iterations is met. As aforementioned, an individual crow might have the best hiding place (memory) at every iteration. Thus, at the maximum iteration, the best memory for the whole swarm is reported as a global optimum solution.

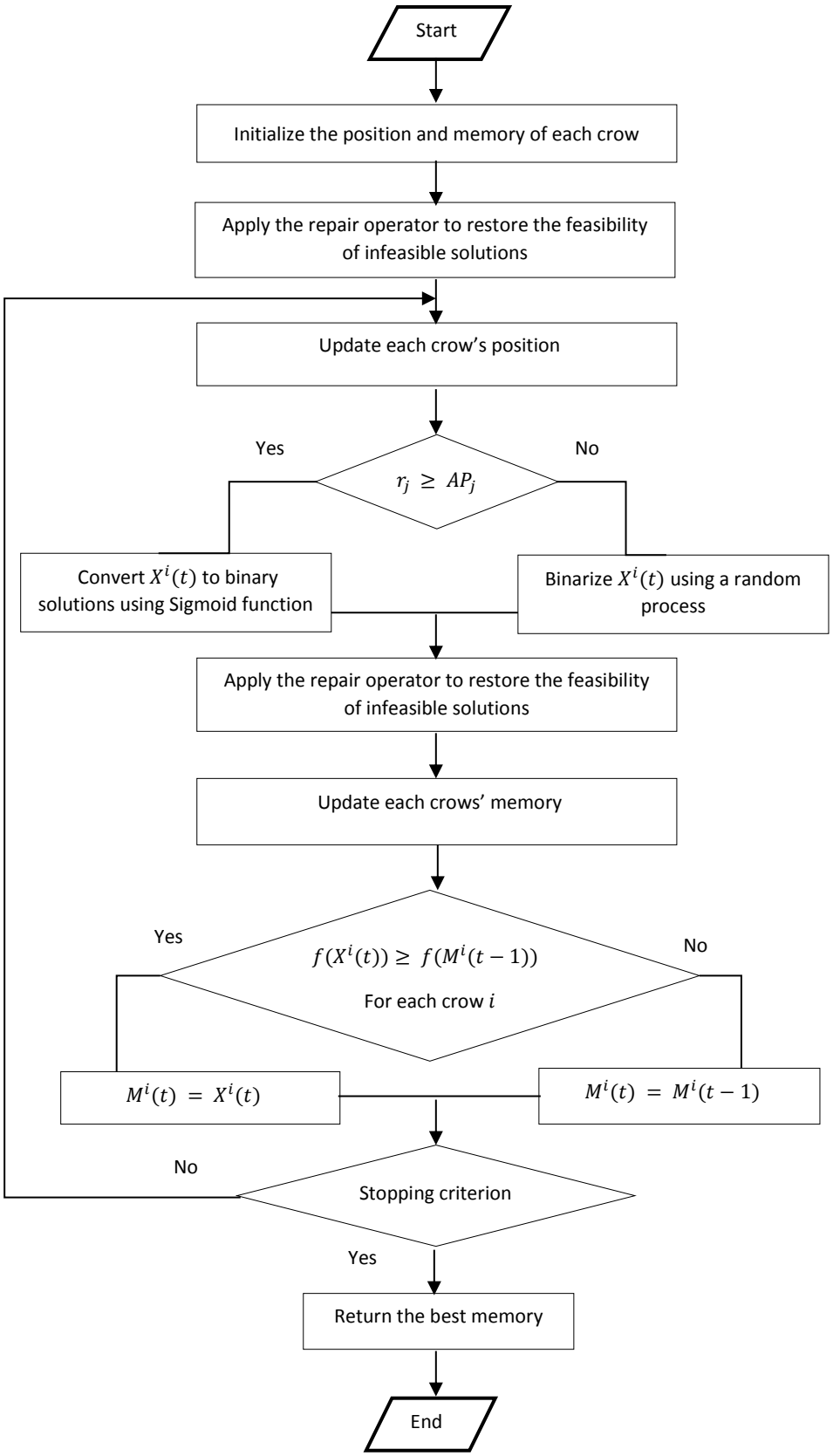


Figure 30. The flowchart of BCSA

6 Experimental results

6.1 Test instances

In order to verify the effectiveness of BCSA, this latter is tested on 2D-BPP benchmarks available at the URL[29] below. In fact, the 2D-BPP benchmark instances are categorized into 10 classes. The first six classes are introduced by (Berkey & Wang, 1987) and the last four classes are proposed by (Martello & Vigo, 1998) including more realistic instances. Each class has a fixed and identical dimension of bins which are 10×10, 30×30, 40×40, 100×100, and 300×300, while the height and the width of each item are randomly generated following a uniform distribution in a specific interval. Each class is divided into five subclasses according to the number of items to be packed (n = 20, 40, 60, 80, 100). Each subclass contains ten problem instances. So, the benchmark instances contain globally 500 problem instances. The obtained classes are built as below:

- The six classes proposed by (Berkey & Wang, 1987) are generated as following:
  - Class I: w_j and h_j uniformly random in [1,10], $W = H = 10$
  - Class II: w_j and h_j uniformly random in [1,10], $W = H = 30$
  - Class III: w_j and h_j uniformly random in [1,35], $W = H = 40$
  - Class IV: w_j and h_j uniformly random in [1,35], $W = H = 100$
  - Class V: w_j and h_j uniformly random in [1,100], $W = H = 100$
  - Class VI: w_j and h_j uniformly random in [1,100], $W = H = 300$
- The four classes proposed by (Martello & Vigo, 1998) are randomly generated so that the items have different sizes. The size of bins is identical for all instances (100×100). Here, four types of items are considered:
  - Type 1: w_j uniformly random in $[2/3\ W, W]$ and h_j uniformly random in $[1, 1/2\ H]$
  - Type 2: w_j uniformly random in $[1, 1/2\ W]$ and h_j uniformly random in $[H, 2/3\ H]$
  - Type 3: w_j uniformly random in $[1/2W, W]$ and h_j uniformly random in $[1/2H, H]$
  - Type 4: w_j uniformly random in $[1, 1/2\ W]$ and h_j uniformly random in $[1, 1/2H]$

6.2 Experimental design

For simulation experiments, one problem instance is randomly selected from each subclass of the above-mentioned instances, considering it as a sufficiently representative instance, which gives a total of 50 test instances instead of 500 test instances.

Furthermore, the proposed BCSA is compared against two algorithms, namely the binary particle swarm optimization algorithm (BPSO) of (Kennedy & Eberhart, 1997), which has

[29] http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem.

undergone some modifications to adapt it to the context of 2D-BPP, and the well-known Finite First Fit heuristic (FFF) of (Berkey & Wang, 1987). All these algorithms are implemented in JAVA programming language using a PC with Intel® Core™ i5, 2.5 GHz, and 4.0 Go of RAM running on 64-bits Windows 7 operating system. Based on our extensive numerical experience, the control parameters in BCSA are set for all runs to values presented in Table 21. However, for BPSO algorithm, we set the parameters to values previously used by other authors.

Table 21. Parameters setting

Algorithms	Parameters	Value
BCSA	Flock size	50
	Flight size	2
	Awareness probability	0,1
	Maximum iteration	100
BPSO	Particle swarm size	50
	Maximum / minimum inertia weight	w_{min} = 0,4; w_{max}= 0,9
	Acceleration coefficients	c_1 = c_2 = 1,5
	Velocity	V_{min} = -2; V_{max} =2
	Maximum iteration	100

6.3 Experimental results

Due to the stochastic nature of BCSA and BPSO, 20 independent runs were executed. The comparison is investigated based on two performance measures: Average fitness value (AVG) and running time (CPU) expressed in seconds. Tables 22 and 23 summarize the comparison of BCSA against FFF and BPSO. In each table, the first and second columns indicate, respectively, the problem class and its respective bin dimensions. The next pair of columns report, respectively, the selected test instances and their corresponding number of items, while the last columns report the computational results of the proposed algorithm and its rival methods.

The experimental results show that BCSA performs much better than both algorithms in terms of AVG in all instances (bold values indicate the best values among algorithms). In other words, BCSA exploits more effectively the used bins (it places the items in a tight space), which allows reducing the number of bins to be used for packing all available items. As far as the runtime is concerned, BCSA needs a few times to find the best solutions in comparison to BPSO. However, FFF yields good results but not better than those of BCSA in extremely rapid time. As can be observed from tables, the BCSA yields better results independently of the number of items and the capacity of bins.

Table 22. Comparison of BCSA results by using Berkey and Wang instances.

Problem class	$H \times W$	N° of instance	n	FFF		BPSO		BCSA	
				CPU	AVG	CPU	AVG	CPU	AVG
Class I	10 × 10	Inst8	20	0,0008	419,69	0,6185	389,04	0,0289	**554,94**
		Inst10	40	0,0000	3442,92	3,2768	3117,98	0,2076	**4788,42**
		Inst5	60	0,0000	13133,92	8,1276	12141,97	0,8858	**16948,96**
		Inst4	80	0,0008	36814,28	16,9325	33332,82	2,3750	**47425,40**
		Inst2	100	0,0008	73044,58	32,4253	67318,06	6,4151	**95381,72**
Class II	30 × 30	Inst1	20	0,0000	10,37	0,5795	9,48	0,0253	**17,99**
		Inst3	40	0,0000	93,92	2,3564	87,78	0,2192	**175,12**
		Inst6	60	0,0008	201,26	8,1266	186,12	1,6102	**274,21**
		Inst7	80	0,0008	684,73	20,0036	634,15	2,9041	**973,27**
		Inst2	100	0,0016	1119,59	32,7567	1044,55	6,3869	**1709,18**
Class III	40 × 40	Inst4	20	0,0000	170,32	0,8095	151,36	0,0262	**268,33**
		Inst6	40	0,0004	1671,88	2,4936	1540,05	0,2132	**2214,68**
		Inst3	60	0,0005	7721,35	7,4652	6956,18	0,9835	**9551,37**
		Inst9	80	0,0016	21299,40	21,6988	19574,45	2,8330	**33326,25**
		Inst8	100	0,0031	36804,78	33,6686	34175,16	6,3895	**50155,57**
Class IV	100 × 100	Inst5	20	0,0000	6,09	0,5546	5,80	0,0336	**12,22**
		Inst8	40	0,0000	99,43	2,4059	92,23	2,4415	**200,20**
		Inst3	60	0,0008	228,21	7,5745	212,02	1,4555	**347,19**
		Inst10	80	0,0016	529,44	16,6386	497,30	2,6485	**1029,12**
		Inst4	100	0,0031	916,19	33,8111	855,79	6,0642	**1538,55**
Class V	100 × 100	Inst10	20	0,0008	513,44	0,4360	494,87	0,0310	**551,35**
		Inst2	40	0,0001	2662,12	2,2714	2435,58	0,2069	**3141,19**
		Inst8	60	0,0002	12323,78	6,8261	11520,90	0,8137	**18568,14**
		Inst1	80	0,0005	23701,70	16,7183	21225,96	2,6943	**35776,63**
		Inst7	100	0,0006	44371,01	33,3851	40885,44	5,9294	**55431,63**
Class VI	300 × 300	Inst5	20	0,0000	4,70	0,4415	4,49	0,0321	**9,63**
		Inst6	40	2,0000	37,84	2,3004	35,74	0,2427	**70,63**
		Inst3	60	0,0000	175,10	6,5669	163,43	0,8489	**216,42**
		Inst4	80	0,0008	406,93	20,4363	381,47	2,8452	**578,74**
		Inst2	100	0,0000	833,20	31,5641	782,07	6,1759	**1160,02**

Table 23. Comparison of BCSA results by using Martello and Vigo instances.

Problem class	$H \times W$	N° of instance	n	FFF		BPSO		BCSA	
				CPU	AVG	CPU	AVG	CPU	AVG
Class VII	100 × 100	Inst5	20	0,0000	359,66	0,4416	262,12	0,0351	**513,20**
		Inst10	40	0,0008	3325,23	2,7616	2515,77	0,1455	**3547,29**
		Inst2	60	0,0000	7764,81	8,2844	5993,76	0,9960	**9189,37**
		Inst9	80	0,0008	25174,89	20,7807	19563,61	3,0081	**31351,27**
		Inst3	100	0,0003	36327,73	43,4908	26891,87	8,1283	**46705,27**
Class VIII	100 × 100	Inst7	20	0,0000	201,20	0,5484	194,47	0,0240	**452,12**
		Inst2	40	0,0000	3625,57	2,7184	3534,28	0,2202	**4779,44**

		Inst9	60	0,0000	9303,43	7,3168	9149,17	0,8273	**15102,90**
		Inst4	80	0,0008	19884,02	19,9404	19359,51	2,5199	**30795,66**
		Inst10	100	0,0008	59385,17	33,9708	58345,50	5,9369	**87019,54**
Class IX	100 × 100	Inst1	20	0,0000	1164,18	0,4480	1164,18	0,0234	**1979,89**
		Inst8	40	0,0000	7591,02	2,2238	7455,61	0,2186	**13881,30**
		Inst3	60	0,0000	27988,11	6,7419	26690,61	0,8955	**49495,61**
		Inst9	80	0,0000	55743,95	20,3263	53357,95	2,5225	**84037,03**
		Inst4	100	0,0008	119126,66	32,6363	113640,01	6,5365	**160066,32**
Class X	100 × 100	Inst10	20	0,0000	72,62	0,4505	68,61	0,0211	**150,58816**
		Inst5	40	0,0000	935,21	2,3374	884,41	0,2318	**1048,6964**
		Inst7	60	0,0000	4338,71	8,0474	4121,32	0,9509	**7150,3205**
		Inst1	80	0,0000	9885,01	19,8340	9155,39	2,5715	**16879,532**
		Inst8	100	0,0008	8225,35	32,7937	7664,58	6,0092	**11745,764**

Figures 31-33 depict a comparison test of BCSA against BPSO and existing lower bounds (LB*) in the literature (see the link[30] below). The comparison is performed based on the average number of bins used in each instance (see the plot on the left) and the time required for each instance to get the best solution (see the plot on the right). The experiments are tested on class IX (Martello & Vigo, 1998). The graph on the left in each figure is plotted with the average used bins as the horizontal axis and the ten test instances of each sub-class as the vertical axis, while the graph on the right is plotted with the average running time as the horizontal axis and the ten test instances as the vertical axis. All the test problems are evaluated over 20 independent runs, and the average results are considered for graphic illustration.

From Figures 31-33, we can clearly see that BCSA yields the best results so near to lower bounds and sometimes better than them. It is also seen that our proposed binary algorithm needs a few bins to pack all items in comparison with BPSO and takes less time than it regarding all test instances.

[30] http://or.dei.unibo.it/general-files/best-known-solution-and-lower-bound-each-instance.

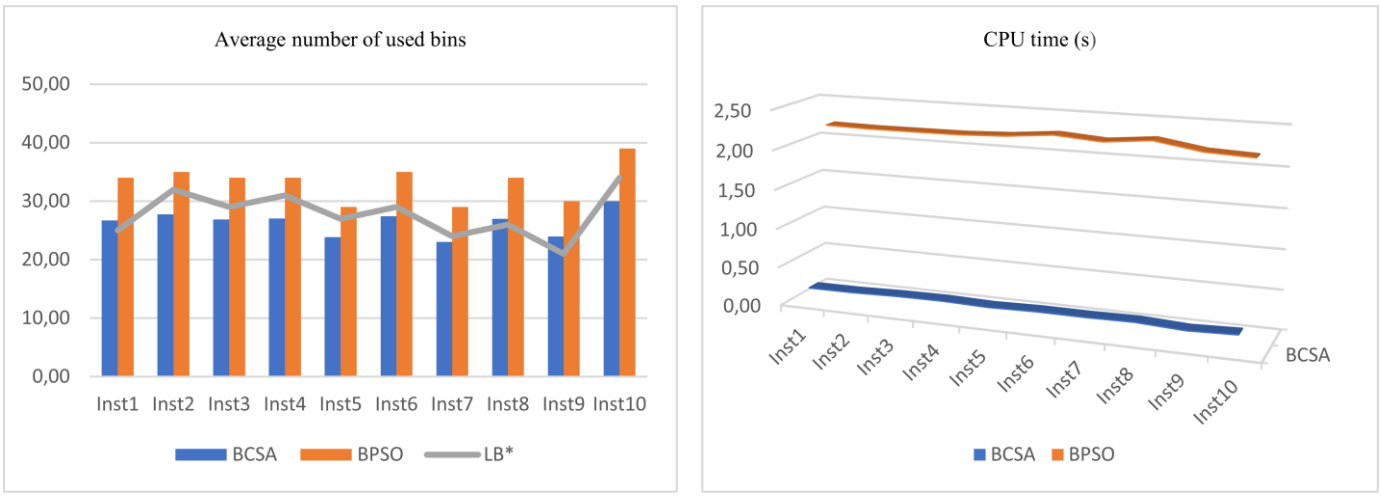


Figure 31. Comparison of BCSA (n = 40)

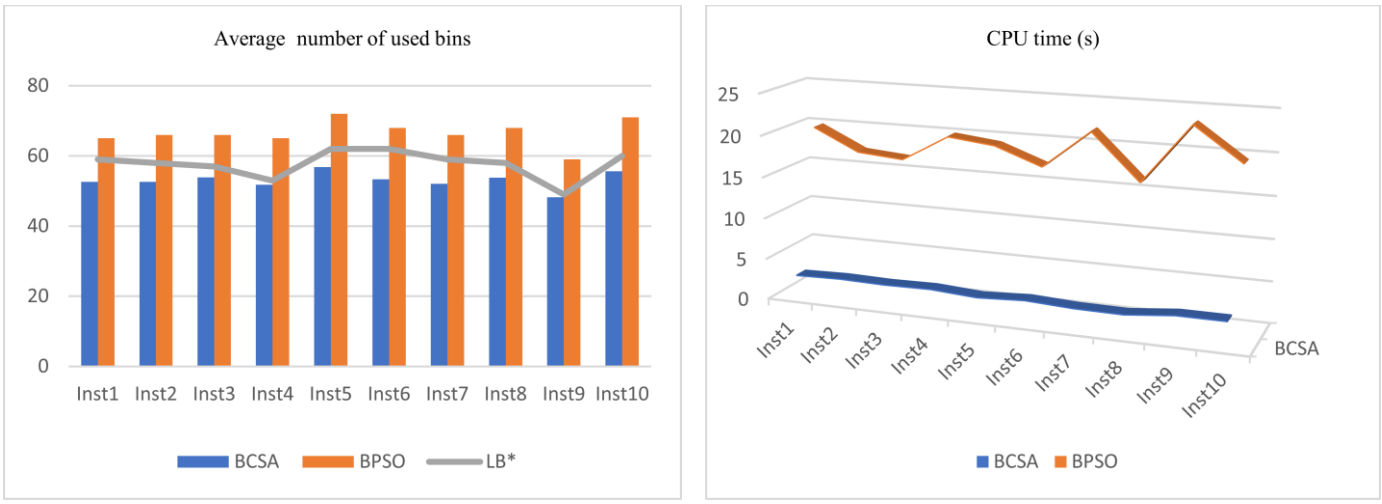


Figure 32. Comparison of BCSA (n = 80)

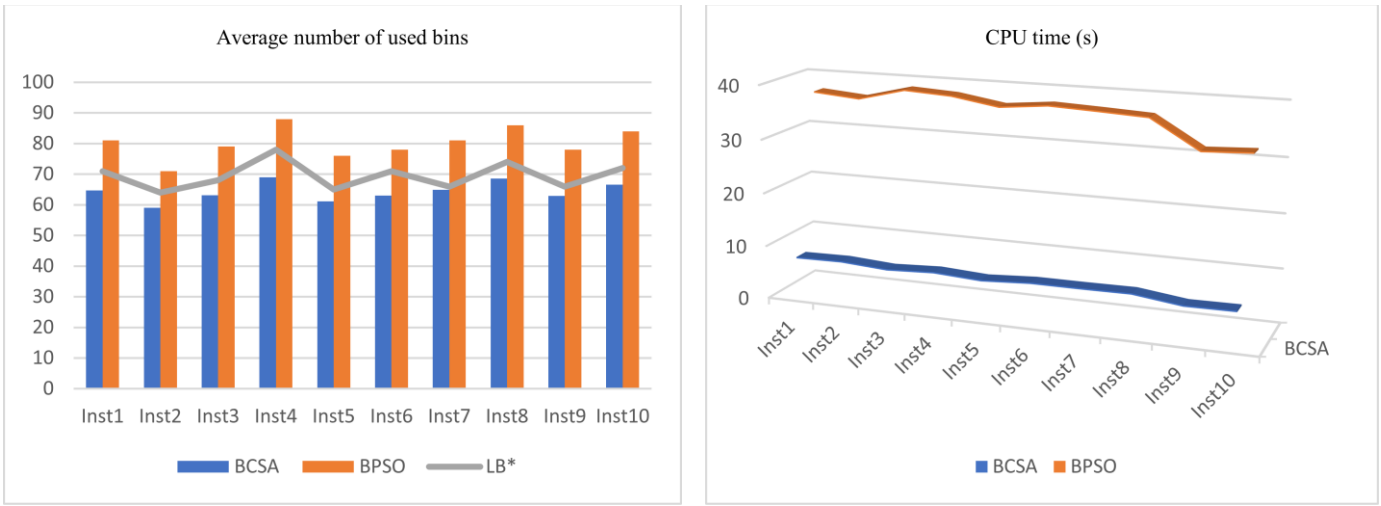


Figure 33. Comparison of BCSA (n = 100)

7 Conclusion

The CSA has a simple mechanism for optimization with few controlling parameters to adjust, which are the flight length and awareness probability. Moreover, CSA is easier to be implemented which makes it very desirable for solving complex optimization problems. In order to adapt the CSA to the binary search space of 2D-BPP, we proposed a binary version of CSA that we called BCSA. In original CSA, each crow either moved toward the best position discovered by another randomly selected crow or to a new random position, while in BCSA the trajectories of crows are changes in the probability that a coordinate will take on zero or one value.

To assess the performance of BCSA, it was tested on 2D-BPP benchmark instances. The simulations have demonstrated that the BCSA can achieve better results in comparison to a greedy heuristic and a swarm intelligent algorithm, this is due mainly to its capability to memorize the best positions and also to the awareness probability and the flight length of crows that have a major role in balancing between diversification and intensification strategies.

Chapter 7: A crow search based genetic algorithm for solving 2D-BPP

1 Introduction

As mentioned before, in order to benefit from the advantages of CSA in solving combinatorial optimization problems, the CSA should be binarized to work efficiently in a binary search space. In this regard, there are various binarization techniques one of them is presented in the previous chapter. However, there is another way more simple and more fruitful, which is to hybridize the CSA with another algorithm working in binary search space. To do that, we have chosen the standard genetic algorithm for three reasons: It is able to be adapted to any kind of optimization problems (discrete, binary, or continuous optimization); It was proved that it is a competitive algorithm for solving 2D-BPP. Thus, if we combine it with CSA, it can either improve the performance of CSA or improved by CSA to finally yield best results; The last reason is that the genetic operators can be easily incorporated in CSA since there is a selection step in CSA that corresponds to selection operation in GA and the process of updating positions of crows that corresponds to variation operators in GA.

The performance of the proposed hybrid approach is evaluated on standard benchmark instances of 2D-BPP and compared with two other bio-inspired algorithms having a closely similar nature, namely the standard GA and the BPSO algorithm. The outcome results are very promising. The remainder of this chapter is organized as follows. An overview of the existing hybrid approaches that combine CSA with other algorithms is provided in section 2. Then, the proposed CSA based on genetic operators is described in section 3. Thereafter, the results reached by the proposed hybrid approach are presented and analyzed in section 4. Finally, some conclusions are given in section 5.

2 Hybrid crow search algorithm

Since the CSA can efficiently solve complex optimization problems, the authors hybridize it with other algorithms in order either to minimize the adjustable parameters of a given algorithm or to benefit from its efficiency and simplicity to solve such problems. You found below some works of researchers that have combined CSA with other approaches hoping to maximize the performance of their proposed algorithms.

(Turgut & Turgut, 2017) have developed a hybrid approach based on CSA principles for solving a design problem, called the counter flow wet-cooling tower optimization problem. They have incorporated the CSA into artificial cooperative search (ACS) algorithm in order to speed up the convergence rate and increase the diversity of solutions. To validate the performance of their hybrid approach, fourteen benchmark test functions were used as well as six variations of the counter flow wet-cooling tower optimization problem were solved having different parameter values. The simulations proved the efficiency of the proposed hybrid algorithm in comparison with three existing algorithms including the ACS algorithm.

(Marichelvam & Geetha, 2018) have proposed a hybrid CSA for solving the flow shop scheduling problems by incorporating the dispatching rules in the CSA process that are the shortest processing time and the longest processing time[31]. An evaluation of their proposed algorithm is carried out on industrial scheduling problems, indicating that the hybrid CSA performs well against many dispatching rules and constructive heuristics. Additionally, empirical experiments were presented for randomly generated problems, which has confirmed the superiority of their hybrid approach in terms of accuracy and convergence speed.

(Allaoui, et al., 2018) have made use of CSA for solving the DNA fragment assembly problem. They have combined CSA with a local search method in order to accelerate the search process and improve the accuracy of solutions. The hybrid optimizer is compared favorably with other existing algorithms.

(Lakshmi, et al., 2018) have integrated CSA in the *k*-means clustering algorithm to escape from local optimum solutions in the selection phase of initial centroids. To investigate the performance of their algorithm, the computational experiments were conducted on benchmark datasets and the outcome results indicated that the proposed hybrid approach outperforms various clustering algorithms and optimization-based clustering algorithms.

(Pasandideh & Khalilpourazari, 2018) have developed an algorithm combining sine cosine algorithm with CSA. The hybrid approach was tested on seven well-known benchmark functions and the results showed that the proposed hybrid algorithm is a competitive optimization tool in comparison with other state-of-the-art metaheuristics.

(Huang & Wu, 2018) have proposed a hybrid approach based on PSO algorithm and CSA. Besides, a local search method has been incorporated into their hybrid algorithm hoping to enhance the quality of solutions. Six benchmark test functions were used in the simulations that have demonstrated the best performance of their algorithm against other approaches in terms of solutions' quality and computing time.

Very recently, (Arora, et al., 2019) have suggested an algorithm hybridizing grey wolf optimizer (GWO) with CSA principles in order to overcome the limitations of GWO algorithm. It was applied to solve function optimization problems as well as feature selection problems. The evaluation of the hybrid approach was carried out upon 23 benchmark test functions having various dimensions and complexities, as well as 21 widely employed data sets for feature selection problems. Thus, the simulations demonstrated that the algorithm outperforms other comparative methods, including the recent variants of GWO algorithm that are the enhanced grey wolf optimizer algorithm and the augmented grey wolf optimizer algorithm, in terms of quality of solutions and consuming time.

[31] The shortest processing time rule orders the jobs in the order of increasing processing times, while the longest processing time rule orders the jobs in the order of decreasing processing times.

3 The crow search based genetic algorithm

In this section, we propose a hybrid approach combining GA with CSA to deal with 2D-BPP, which we call Crow Search-based Genetic Algorithm (CSGA). We are first motivated by the smart behavior of crows in CSA and then by the evolutionary behavior of individuals in GA. In comparison with GA, in CSGA the whole population shares its information through the crossover operation, and hence neither the crossover probability is considered nor a replacement strategy is needed to create the upcoming generations. In comparison with CSA, in CSGA each crow *i* selects one crow from the swarm to follow it by using a selection operator. In addition, a mutation operator is applied with a mutation probability hoping to increase the population diversity. The following subsections will elaborate on the process of CSGA.

3.1 Building the initial population

In our representation schema, we have chosen the same encoding as in the previous chapter. For recalling, the positions of crows in the d-dimensional environment are generated by allocating or not an item to a given bin. Each position is considered as a potential solution of the problem. It is encoded as a $n \times d$ matrix, where n is the number of items to be packed into bins and d is regarded as the number of available bins. The representation of each solution i, in a population having N crows, is shown below:

$$X^i = \left(x_{jk}^i\right) \in \{0,1\}^{n \times d}; \quad for\ i \in \{1, \dots, N\} \tag{7.1}$$

For all i, the decision variable x_{jk}^i takes the value 1 if the j-th item is packed in the k-th bin and it is equal to 0 otherwise. For the sake of clarity, an item is allocated to one and only one bin, and then it is immediately removed from the list of items. Notice that the packing of each item must respect the Bottom-Left strategy.

Moreover, the memory matrix M^i has the same dimensions as the position matrix X^i of each crow i. In the initial stage, the position matrix coefficients are randomly generated and the memory matrix of each crow i coincides with the initial position matrix.

3.2 Checking the feasibility of solutions

The random process at the initial stage can generate infeasible solutions. You found below the capacity constraint that should be respected in each bin:

$$\text{For } i \in \{1, \dots, N\}: \ \sum_{j=1}^{n} w_j \times h_j \times x_{jk}^i \leq W \times H \times y_k; \ \forall k \in \{1, \dots, d\} \tag{7.2}$$

Where,

$$y_k = \begin{cases} 1 & if \ \ the\ bin\ k\ is\ used \\ 0 & otherwise \end{cases} \tag{7.3}$$

Notice that the bin k is considered as a used bin, if $\sum_{j=1}^{n} x_{jk}^{i} \geq 1$, *i.e.* it contains at least one item, while it is not considered as a used bin whether $\sum_{j=1}^{n} x_{jk}^{i} = 0$. However, to deal with infeasible solutions, the specific-knowledge repair operator proposed in the previous chapter is applied.

On the other side, we make use of the following fitness function to assess the accuracy of potential solutions:

$$f = \frac{1}{\sum_{k=1}^{d} y_k} \tag{7.4}$$

Equation (7.4) corresponds to the reverse of the objective function of 2D-BPP, which means that the fewer the bins used to pack all items, the higher the fitness value.

3.3 Updating crows' positions and memory

The GA operators are combined with the CSA process in order to update the crows' positions. Henceforth, we consider crow positions as chromosomes. Each column of the position matrix corresponds to a gene of the chromosome, while each gene is encoded by a binary vector of n alleles. Figure 34 shows an example of a chromosome representation schema.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 34. Representation of a chromosome with (n = 5) and (d =4)

Note that the number of bins is unlimited, but we assume in the worst case that one item can be packed in one bin. Thus, as a first step, the number of bins is equal to the number of available items and then this number must be optimized.

3.3.1 The selection operation

In this stage, each crow i selects a crow j from the population by using a binary tournament selection. In fact, the binary tournament selection (BTS) consists of choosing at random two crows from the population and making a competition between them. The winner is a crow with the highest fitness value. So, the crow i chooses the winner as a target to follow, which allows to update its current position.

3.3.2 The crossover operation

For reminding, the crossover is the process of exchanging genes between individuals in order to create new ones. Its main goal is to exploit more the search space of solutions. In this

stage, we combine the CSA rules with a crossover strategy in order to update positions as follows: If $r_j \geq AP_j$, we cross the position of crow i with the position of the selected crow j. Else, the position of the crow i is crossed with its memory. A PMX crossover operator, developed by (Goldberg & Lingle, 1985), is applied to recombine two positions or even the position and memory. The example in Figure 35 explains how we adapt it in our context. Indeed, we choose at random a bin from the first parent position (*e.g.* bin 1) and we replace its content {item 2, item 3} in the offspring at the bin with the same label. The other bins inherit their contents from the second parent. The duplicate items are replaced by items packed in bin 1 of the second parent while respecting the same order. In our example, items 2 and 3 are packed respectively in bins 3 and 2 of the second parent. So, we replace item 2 by item 4 and item 3 by item 5. Note that the sign «⊗» indicates the crossover operation.

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Figure 35. An example of the crossover operation with (n=5) and (d =4)

Once the crossover strategy has been accomplished, the offspring positions are evaluated. If it is necessary, the repair operator described in the previous chapter is applied for restoring the feasibility of solutions.

3.3.3 The Mutation operation

In addition to crossover operation, the mutation operator is used to explore more regions of solution search space hopping to find new better solutions. We mutate some generated offspring positions by using the split bin mechanism proposed by (Liu, et al., 2006). It consists of choosing randomly one used bin and split its content into two bins. In our algorithm, we have applied this mechanism while making some changes according to the following manner. The first part of items is kept in its original bin and the second part of items is merged into other bin that can accommodate it without violating the constraint capacity. If no used bin can pack it, a new one is created.

3.3.4 Updating crows' memory

After mutation strategy, the quality of each offspring position is investigated using the fitness function and then compared with the previous memory. If the offspring position is better than the recently memorized position, then the crow updates its memory with the offspring position. Otherwise, it keeps its previous memory.

3.4 The framework of CSGA

The overall algorithm is summarized in Figure 36.

Firstly, the CSGA initializes the population with N crows and sets the control parameters such as the maximum number of generations G, the flight length, and the awareness probability of crows. Besides, the CSGA sets the initial memory of each crow i as below:

$$M^i(G = 1) = X^i\,(G = 1) \tag{7.5}$$

Then the hybrid approach evaluates the feasibility of solutions and calculates their fitness values. For each infeasible solution, it applies the proposed repair operator.

For the next step, it applies a binary tournament selection to select a crow j to be followed by a crow i. To update the position of each crow i, it makes use of the following equations:

$$\text{If} \quad r_j \geq AP_j, \quad \text{then} \;\; X^i(G+1) = X^i(G) \otimes X^j(G) \tag{7.6}$$

$$\text{Else, } X^i(G+1) = X^i(G) \otimes M^i(G) \tag{7.7}$$

Also in this step, the repair operator is applied to restore the feasibility of generated offspring positions that violate the capacity constraints.

After crossing positions, the CSGA could apply the mutation strategy to some resulting offspring positions with a preset probability p_m. Then, it calculates the fitness function for each offspring position and updates the memory of each crow by the same equation as in the previous chapter (see equation 6.12).

All these steps are repeated until a maximum number of generations and then CSGA outputs the best found memory in the entire population as an optimum solution of 2D-BPP.

.

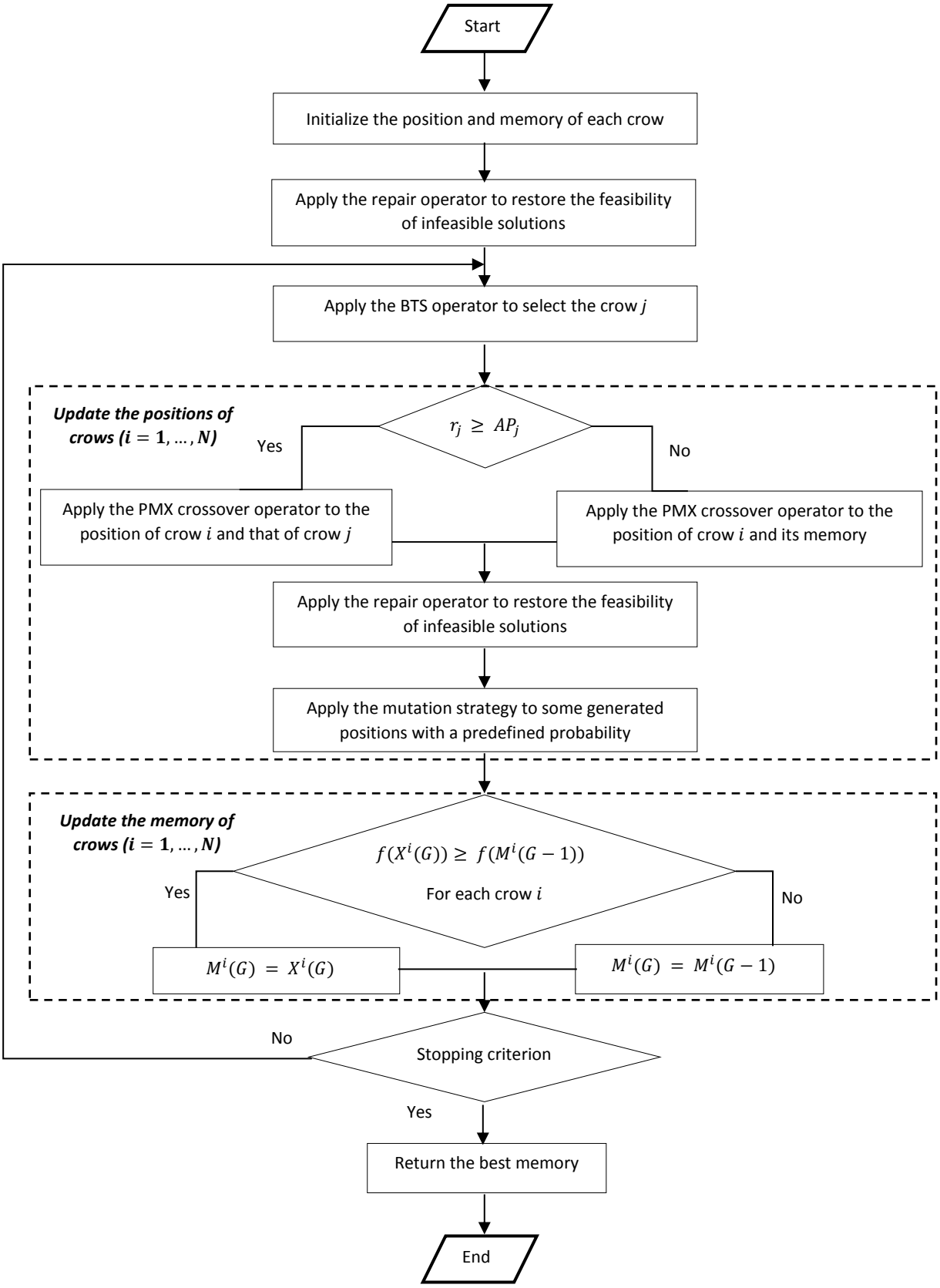


Figure 36. The flowchart of CSGA

4 Experimental results

4.1 Experimental design

The performance of CSGA is examined on various test instances of 2D-BPP having different sizes and varied complexities. It should be emphasized that the considered test data are the same ones described in the previous chapter. We recall that there are 10 classes randomly generated by varying the capacity of bins and the range of items' size. The first six classes were introduced by (Berkey & Wang, 1987), while the last four classes were proposed by (Martello & Vigo, 1998). During the experiments, only four problem classes have been investigated (see Table 24): Three representative classes from Berkey and Wang classes (we have chosen only one class from two classes having the same size distribution of items) and another one representative class from Martello and Vigo classes. However, in each selected class, we have considered just three values of the number of items that are $n = 20$, $n = 80$, and $n = 100$. This choice is due to the fact that these instances are more representative, *i.e.* an instance with $n = 20$ represents small size problem, an instance with $n = 80$ represents medium size problem and an instance with $n = 100$ represents large size problem. Notice that the proposed CSGA is coded in JAVA on a PC having an Intel Core i5 with 2.5 GHz and 4.0 GB of RAM, running on 64-bits Windows 7 operating system.

Table 24. Information about the test instances used in experiments

Problem class	Bin capacity	Item height	Item width
Class I	$W = H = 10$	[1, 10]	[1, 10]
Class III	$W = H = 40$	[1, 35]	[1, 35]
Class V	$W = H = 100$	[1, 100]	[1, 100]
Class IX	$W = H = 100$	70% in $[1/2H, H]$ and 10% in $[2/3H, H]$, $[1, 1/2H]$	70% in $[1/2W, W]$ and 10% in $[2/3W, W]$, $[1, 1/2W]$

To validate the efficiency of CGSA, this latter is compared against two bio-inspired algorithms: The BPSO (Kennedy & Eberhart, 1997) and the standard GA. The BPSO has undergone minor modifications in order to adapt it to the 2D-BPP context, especially, in encoding schema and in the fact of incorporating a repair operator in order to meet the capacity constraint of bins. The standard GA applies the same genetic operators like the ones used by CSGA (see section 4). The control parameters of each algorithm are reported in Table 25. For clarity, the *Popsize* means the size of population, *Maxgen* indicates the maximum number of generations, w_{min} and w_{max} denote respectively the minimum and maximum inertia weight, additionally, c_1 and c_2 denote the acceleration coefficients. Finally, V_{min} and V_{max} refer to the minimum and maximum velocity of particles. It is worth mentioning that the control parameters of CSGA are set based upon many independent experiences, while the BPSO parameters are chosen according to the experience of other authors.

Table 25. Parameters setting

Algorithms	Parameter values
CSGA	*Popsize* = 100; *AP* = 0,01; p_m = 0,10; *Maxgen* = 100
GA	*Popsize* = 100; p_c = 0,95; p_m = 0,01; *Maxgen* = 100
BPSO	*Popsize* = 100; w_{min} = 0,4; w_{max} = 0,9; $c_1 = c_2$ = 1,5; V_{min} =-2; V_{max} = 2; *Maxgen* = 100

4.2 Experimental results

All test cases of CSGA as well as the comparative algorithms were evaluated through 30 independent runs and then the average results are retained. To assess our results, in terms of fitness value, we applied the Wilcoxon test that is used in order to determine whether the average fitness values of CSGA compared to those of BPSO and GA are significantly different or not. We note that we have used the SPSS statistics 22 for statistical testing. The significance level is fixed at 0.05, if the p-value is less than 0.05, we reject the null hypothesis that assumes there is no significant difference between two compared algorithms. Otherwise, we accept the alternative hypothesis that assumes there is a significant difference between them. The obtained results of the p-value are shown in Table 26. For recalling, the value of R- (resp. R+) in Table 26 indicates the sum of the ranks corresponding to negative (resp. positive) difference between the results of two algorithms. The value of s in Table 26 represents the statistical result of a pairwise comparison: $s = 1$ indicates that the first algorithm is significantly better than the latter; $s = -1$ indicates that the first algorithm is significantly worse than the latter. Both values -1 and 1 mean there is a significant difference. However, $s = 0$ indicates that there is no significant difference between the two compared algorithms. From Table 26, we can clearly observe that CSGA outperforms the standard GA, while it is sometimes nearly equivalent to BPSO and sometimes it is outperformed by BPSO.

Table 26. Wilcoxon test on the average fitness value of CSGA against other algorithms

Problem class	Number of items	Compared algorithms	R+	R-	p-value	s
Class I	20	CSGA - GA	55	0	0,005	1
		CSGA - BPSO	18,5	2,5	0,093	0
	60	CSGA - GA	55	0	0,005	1
		CSGA - BPSO	27,5	27,5	1,000	0
	100	CSGA - GA	55	0	0,005	1
		CSGA - BPSO	55	0	0,005	1
Class III	20	CSGA - GA	55	0	0,004	1
		CSGA - BPSO	2	13	0,136	0
	60	CSGA - GA	55	0	0,005	1
		CSGA - BPSO	0	55	0,005	-1
	100	CSGA - GA	55	0	0,005	1
		CSGA - BPSO	2	53	0,009	-1
	20	CSGA - GA	55	0	0,005	1

		CSGA - BPSO	4	41	0,028	-1
Class V	60	CSGA - GA	55	0	0,005	1
		CSGA - BPSO	0	55	0,005	-1
	100	CSGA - GA	55	0	0,005	1
		CSGA - BPSO	0	55	0,005	-1

On the other side, Tables 27-29 report the empirical results of CSGA, GA, and BPSO in terms of average used bins obtained over 30 runs and the average execution time. These tests are performed in class IX. In each table, the first column indicates the test instances of class IX. The second column show the benchmark lower bounds of 2D-BPP presented in the link[32] below. The last columns record the obtained average results of the three compared algorithms.

From Tables 27-29, we can see that the number of used bins obtained by CSGA, in each instance, is near or equal to the corresponding lower bound and sometimes even better than the latter. Furthermore, the proposed hybrid algorithm yields better results in less computational time in comparison with both BPSO and GA. We note that the best results are in bold and the computation time is expressed in seconds.

Table 27. Comparative results of CSGA versus GA and BPSO using instances with 40 items

Test instance	LB*	CSGA		GA		BPSO	
		Used bins	CPU time	Used bins	CPU time	Used bins	CPU time
Instance1	25	**29**	**0,067**	37	2,013	34	4,346
Instance2	32	**29**	**0,066**	36	1,916	35	4,383
Instance3	29	**29**	**1,914**	35	**1,914**	34	4,386
Instance4	31	**28**	**0,063**	35	2,085	34	4,472
Instance5	27	**27**	**0,064**	33	1,993	29	4,382
Instance6	29	**29**	**0,067**	36	2,030	35	4,393
Instance7	24	**26**	**0,068**	32	1,920	29	4,421
Instance8	26	**28**	**0,063**	35	2,006	34	4,676
Instance9	21	**25**	**0,062**	33	1,995	30	4,389
Instance10	34	**30**	**0,064**	38	2,028	39	4,423
Average	27,8	**28**	**0,250**	35	1,990	33,3	4,427

Table 28. Comparative results of CSGA versus GA and BPSO using instances with 80 items

Test instance	LB*	CSGA		GA		BPSO	
		Used bins	CPU time	Used bins	CPU time	Used bins	CPU time
instance1	59	**60**	**0,264**	70	7,816	65	34,436
instance2	58	**59**	**0,271**	69	7,718	66	39,465
Instance3	57	**59**	**0,252**	70	8,001	66	34,008
Instance4	53	**60**	**0,263**	69	7,753	65	37,740
Instance5	62	**61**	**0,259**	74	7,772	72	32,446
Instance6	62	**59**	**0,251**	71	7,576	68	31,868
Instance7	59	**59**	**0,262**	70	7,833	66	39,240

[32] http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem.

Instance8	58	**58**	**0,264**	71	7,902	68	32,110
Instance9	49	**54**	**0,261**	59	40,232	59	34,454
Instance10	60	**61**	**0,274**	74	7,575	71	32,291
Average	57,7	**59**	**0,262**	69,7	11,018	66,6	34,806

Table 29. Comparative results of CSGA versus GA and BPSO using instances with 100 items

Test instance	LB*	CSGA		GA		BPSO	
		Used bins	CPU time	Used bins	CPU time	Used bins	CPU time
instance1	71	**75**	**0,377**	87	11,755	81	63,273
instance2	64	**69**	**0,371**	86	11,785	81	64,554
Instance3	68	**61**	**0,384**	82	11,767	71	65,076
Instance4	78	**76**	**0,372**	92	11,872	88	62,890
Instance5	65	**73**	**0,388**	84	11,892	76	74,067
Instance6	71	**74**	**0,381**	78	64,425	78	73,664
Instance7	66	**69**	**0,380**	87	12,084	81	64,983
Instance8	74	**73**	**0,375**	90	11,859	86	62,992
Instance9	66	**66**	**0,379**	85	12,039	78	73,892
Instance10	72	**75**	**0,384**	84	64,165	84	65,849
Average	69,5	**71,1**	**0,379**	85,5	22,364	80,4	67,124

4.3 Results analysis

From Table 26, the outcomes of experiments tested on classes I, III, and V have shown that the average results of CSGA are better than GA and worse than BPSO. For these test classes, the difficulty grows quite regularly as the number of available items increases (see the work of (Martello & Vigo, 1998). However, from Tables 27-29, the computations proved that CSGA allocates the available items to the smallest number of bins when compared with GA, BPSO, and even LB*. In class IX, the generated data set seems to be easy than the other three ones, because it is characterized by a high percentage of large items and consequently a few free areas to be managed. Such instances illustrate real-life situations as it is mentioned by (Lodi, et al., 1999).

To summarize, the CSGA has a better performance in comparison with GA independently of whatever the type of instances. Nevertheless, it has generally a worst performance against BPSO in difficult instances (classes I, III, and V), but it performs better than BPSO in more realistic situations as the test instances of class IX in which it could achieve high-quality solutions in a few amount of time.

5 Conclusion

In this chapter, we proposed a hybrid optimizer combining the crow search algorithm with the genetic algorithm for solving 2D-BPP. The main idea behind this hybridization is to discretize and adapt CSA to 2D-BPP context by using genetic operators, on one hand, and to expect reaching a sort of cooperation and synergy between the genetic operators and the CSA

process, on the other hand. We have used various benchmark tests to examine the performance of our hybrid algorithm. Thereafter, we have compared it with two state-of-the-art algorithms that have the same nature (*i.e*. bio-inspired algorithms). The experimental results showed that the proposed approach gives good results against that of GA regardless of the type of instances. However, our algorithm is outperformed by BPSO in difficult instances, but it performs better than BPSO in more realistic instances. In terms of running time, our algorithm is quite fast than both comparative algorithms in easy instances as well as in difficult instances. These results are very encouraging, proving thus the efficiency of the CSGA. As future work, since our algorithm incorporates the genetic operators into the CSA process, it could be applied like GA to solve several optimization problems that work in discrete search space as well as in continuous search space.

Conclusion

The knapsack problem and the bin packing problem are NP-hard combinatorial optimization problems. Such problems occur in different fields like industry, engineering, finance, transportation and logistics, computer networks, and so on. In this thesis, we have focused our attention on generalizing cases of these two problems, which are the 0/1 multidimensional knapsack problem (0/1 MKP) and the two-dimensional bin packing problem (2D-BPP).

For the 0/1 MKP, we have proposed two improved genetic algorithms. The first genetic algorithm concerns a genetic algorithm based on sexual selection, denoted ISGA for Improved Sexual Genetic Algorithm. The sexual selection proposed in this work is an extension of that proposed by (Varnamkhasti & Lee, 2012a) in phenotype space. During our sexual selection strategy, the female chromosomes were selected by binary tournament selection whereas the male chromosomes were selected based on the degree of similarity measured by Manhattan distance, the degree of adaptability measured by the fitness function, or the weight of packed items. To benefit from the characteristics of selected chromosomes, we have applied in crossover operation the so-called two-stage recombination operator while suggesting two versions to adapt it in the context of 0/1 MKP. The experimental results were carried out on widely used benchmark instances that have several sizes and various complexities, indicating that the combination of the two proposed genetic operators in the same framework improves the performance of the genetic algorithm; precisely it prevents the premature convergence by increasing the diversity in the population of solutions. The second developed genetic algorithm consists of incorporating the *k*-means clustering algorithm, especially in selection operation, in order to ensure a high level of genetic diversity at the beginning of the algorithm. The proposed genetic algorithm based *k*-means, which we have called GA-based *k*-means method, firstly divides the population of solutions into *k* (=2) groups based on the similarity of their genetic information. Then, it selects from each cluster one parent. The first parent is randomly selected, while the second one is selected by applying the binary tournament selection. Such a selection strategy allows reproducing non-identical offspring solutions at the crossover stage, thereby preventing the algorithm from being trapped in the local optima. The proposed algorithm was tested on small to large-scaled benchmark instances of 0/1 MKP, and the simulation results proved its efficiency against other genetic algorithms using classical selection operators.

On the other side, we have proposed two swarm intelligence-based algorithms for tackling 2D-BPP. The first algorithm concerns a binary version of a newly developed algorithm called the crow search algorithm (CSA). Indeed, the CSA was originally introduced to deal with continuous optimization problems, proving thus its efficiency and effectiveness in solving this kind of optimization problems. In this thesis, we have binarized the CSA by means of a Sigmoid function. This latter is a straightforward technique of binarization that has successfully applied

to other existing algorithms such as the particle swarm optimization algorithm, the artificial algae algorithm, the flower pollination algorithm, and so on. It determines the probability that a decision variable will take one state or the other (0 or 1 in binary context). Besides, a specific knowledge repair operator is incorporated into our proposed optimizer to deal with infeasible solutions, in one hand, and to increase its accuracy on the other hand. The computational experiments were carried out on a total of 50 benchmark test instances of 2D-BPP. In comparison with two other state-of-the-art algorithms, the outcome results demonstrated that the proposed binary version of CSA, which we have called BCSA, is more robust and achieves better average performance in a reasonable runtime. For the second proposed algorithm devoted to 2D-BPP, it is about a hybrid approach combining CSA principles with the genetic algorithm, denoted CSGA. The main motivation behind such hybridization is adapting the CSA to binary search space of 2D-BPP, also taking profit from the simplicity and the ability of genetic algorithms to resolve different types of optimization problems, and obtaining better performing hybrid approach that exploits and combines advantages of CSA and genetic strategies. To investigate the performance of CSGA, it was tested on 2D-BPP benchmark instances and was compared against two other bio-inspired algorithms. The computational results showed that CSGA is capable of obtaining promising solutions over instances having various sizes and complexities whilst requiring only a modest amount of computational time.

This thesis gives several future work directions. For example, the ISGA may be easily extended by taking into account additional constraints to solve 0/1 MKP variants and its real-life applications. In addition, the ISGA represents a good starting point for developing new specific knowledge genetic operators inspired by the ones presented in this thesis. However, the GA-based *k*-means method can be applied to other optimization problems since it is independent of the structure of the problem at hand. For the two algorithms BCSA and CSGA, they can represent good alternatives for solving other variants of 2D-BPP while taking into account other practical considerations (*e.g.* the guillotine constraint, possibility of rotation items by 90°, the fragility of items…). Furthermore, BCSA can deal with other binary optimization problems by applying some minor modifications like replacing the proposed repair operator by other one specific to the problem at hand…. Moreover, the CSGA seems like a good starting point for developing new genetic operators that take profit from the 2D-BPP structure in order to improve its efficiency. Another perspective is suggesting a new hybrid approach combining CSA with another adequate algorithm (*e.g.* ant colony optimization algorithm, genetic programming, or tabu search…) or using other binarization techniques in BCSA, for adapting them in binary search space and achieving high performance in solving 2D-BPP.

Bibliography

Abdelaziz, A. Y., Ali, E. S. & Elazim, S. A., 2016. Flower pollination algorithm to solve combined economic and emission dispatch problems. *Engineering Science and Technology, an International Journal,* Volume 19, pp. 980-990.

Abdel-Basset, M., El-Shahat, D., El-Henawy, I. & Sangaiah, A. K., 2018. A modified flower pollination algorithm for the multidimensional knapsack problem: human-centric decision making. *Soft Computing,* Volume 22, pp. 4221-4239.

Adhi, A., Santosa, B. & Siswanto, N., 2018. A meta-heuristic method for solving scheduling problem: crow search algorithm. *In IOP Conference Series: Materials Science and Engineering*, April, Volume 337.

Adhi, A., Santosa, B. & Siswanto, N., 2018. A meta-heuristic method for solving scheduling problem: crow search algorithm. *In IOP Conference Series: Materials Science and Engineering*, April, Volume 337.

Aggarwal, C., Hinneburg, A. & Keim, D., 2001. On the Surprising Behavior of Distance Metrics in High Dimensional Space. *Lecture Notes in Computer Science*.

Aghezzaf, B. & Naimi, M., 2009. The two-stage recombination operator and its application to the multiobjective 0/1 knapsack problem: A comparative study. *Computers & Operations Research,* Volume 36, pp. 3247-3262.

Akcay, Y., Li, H. & Xu, S. H., 2007. Greedy algorithm for the general multidimension-al knapsack problem. *Annals of Operations Research,* Volume 150, pp. 17-29.

Akin, M. H., 2009. *New heuristics for the 0-1 multi-dimensional knapsack problems University of Central Florida,* USA: s.n.

Allaoui, M., Ahiod, B. & El Yafrani, M., 2018. A hybrid crow search algorithm for solving the DNA fragment assembly problem. *Expert Systems with Applications,* Volume 102, pp. 44-56.

Amossen, R. R. & Pisinger, D., 2010. Multi-dimensional bin packing problems with guillotine constraints. *Computers & Operations Research,* Volume 37, pp. 1999-2006.

Arango, C., Cortés, P., Escudero, A. & Onieva, L., 2013. Genetic Algorithm for the Dynamic Berth Allocation Problem in Real Time. *In Swarm Intelligence and Bio-Inspired Computation,* pp. 367-383.

Arora, S. et al., 2019. A new hybrid algorithm based on grey wolf optimization and crow search algorithm for unconstrained function optimization and feature selection. *IEEE Access,* Volume 7, pp. 26343-26361.

Arthur, D. & Vassilvitskii, S. .., 2007. k-means++: The advantages of careful seeding. *In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics.*, January, pp. 1027-1035.

Askarzadeh, A., 2016. A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm.. *Computers & Structures,* Volume 169, pp. 1-12.

Atılgan, C. & Nuriyev, U., 2012. Hybrid heuristic algorithm for the multidimensional knapsack problem. *Proceedings of the IEEE 4th International Conference on Problems of Cybernetics and Informatics*, pp. 1-4.

Azad, M., Rocha, A. & Fernandes, E., 2014. Improved Binary Artificial Fish Swarm Algorithm for the 0-1 Multidimensional Knapsack Problems. *Swarm and Evolutionary Computation,* Volume 14, pp. 66-75.

Balas, E., 1965. An Additive Algorithm for Solving Linear Programs with Zero-One Variables. *Operations Research,* Volume 13, pp. 517-546.

Balas, E. & Martin, C. H., 1980. Pivot and complement–a heuristic for 0-1 programming. *Management Science,* Volume 26, pp. 86-96.

Bandyopadhyay, S. & Maulik, U., 2002. An evolutionary technique based on K-means algorithm for optimal clustering in RN. *Information Sciences,* Volume 146, pp. 221-237.

Battiti, R. & Tecchiolli, G., 1995. Local Search with Memory: Benchmarking RTS. *OR Spectrum,* Volume 17, p. 67–86..

Baykasoğlu, A. & Ozsoydan, F., 2014. An Improved Firefly Algorithm for Solving Dynamic Multidimensional Knapsack Problems. *Expert Systems with Applications,* Volume 41, pp. 3712-3725.

Beheshti, Z., Shamsuddin, S. & Yuhaniz, S., 2013. Binary accelerated particle swarm algorithm (BAPSA) for discrete optimization problems. *Journal of Global Optimization,* Volume 57, pp. 549-573.

Berkey, J. O. & Wang, P. Y., 1987. Two-dimensional finite bin-packing algorithms. *Journal of the operational research society,* Volume 38, pp. 423-429.

Berkhin, P., 2006. A Survey of Clustering Data Mining Techniques.. Dans: *Grouping Multidimensional Data.* Berlin: Springer.

Bertsimas, D. & Demir, R., 2002. An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science,* Volume 48, pp. 550-565.

Blum, C. & Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys,* Volume 35, pp. 268-308.

Blum, C. & Schmid, V., 2013. Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm. *Procedia Computer Science,* Volume 18, pp. 899-908.

Bolaji, A. L. A., Babatunde, B. S. & Shola, P. B., 2018. Adaptation of Binary Pigeon-Inspired Algorithm for Solving Multidimensional Knapsack Problem. *Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing,* Volume 583, pp. 743-751.

Bole, A. V. & Kumar, R., 2017. Multidimensional 0–1 Knapsack using directed bee colony algorithm. *In International Conference on Intelligent Techniques in Control, Optimization and Signal Processing*, March, pp. 1-10.

Bonyadi, M. R. & Michalewicz, Z., 2012. A fast particle swarm optimization algorithm for the multidimensional knapsack problem. *In IEEE Congress on Evolutionary Computation*, June, pp. 1-8.

Boussier, S. et al., 2009. Solving the 0-1 multidimensional knapsack problem with resolution search. *arXiv preprint arXiv:0905.0848* .

Boussier, S. et al., 2010. A multi-level search strategy for the 0-1 multidimensional knapsack problem. *Discrete Applied Mathematics,* Volume 158, pp. 97-109..

Boyer, V., 2004. *Méthodes et/ou mixte pour la programmation linéaire en variables 0-1,* Toulouse: s.n.

Boyer, V., Elkihel, M. & El Baz, D., 2009. Heuristics for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research,* Volume 199, pp. 658-664.

Cabot, A., 1970. An enumeration algorithm for knapsack problems. *Operations Research,* Volume 18, pp. 306-311.

Cai, Y., 2010. Artificial fish school algorithm applied in a combinatorial optimization problem. *International Journal of Intelligent systems and applications* , Volume 2.

Cauchy, A., 1847. Méthode générale pour la résolution des systemes d'équations simultanées.. *Comp. Rend. Sci.* , Volume 25, pp. 536-538.

Chen, F., La Porta, T. & Srivastava, M. B., 2012. Resource Allocation with Stochastic Demands. *8th IEEE International Conference on Distributed Computing in Sensor Systems*.

Chih, M., 2015. Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem. *Applied Soft Computing,* Volume 26, pp. 378-389.

Chih, M., Lin, C. J., Chern, M. S. & Ou, T. Y., 2014. Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem. *Applied Mathematical Modelling,* Volume 38, pp. 1338-1350.

Christodoulos, F. & Visweswaran, 1995. quadratic optimization. *Handbook of global optimization,* pp. 217-269..

Chung, F. R., Garey, M. R. & Johnson, D. S., 1982. On packing two-dimensional bins. *SIAM Journal on Algebraic Discrete Methods,* Volume 3, pp. 66-76.

Chu, P. C. & Beasley, J. E., 1998. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics,* Volume 4, pp. 63-86.

Chvátal, V., 1997. Resolution search. *Discrete Appl. Math.,* Volume 73, pp. 81-99.

Clautiaux, F., Carlier, J. & Moukrim, A., 2007. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research,* Volume 183, pp. 1196-1211.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C., 2009. *Introduction to algorithms.* s.l.:MIT press.

Crawford, B. et al., 2017. Putting continuous metaheuristics to work in binary search spaces. *Complexity,* Volume 2017.

D'Ambrosio, C., 2010. *Application-oriented mixed integer non-linear programming,* s.l.: Springer-Verlag.

Dammeyer, F. & Voss., S., 1993. Dynamic Tabu List Management Using Reverse Elimination Method. *Annals of Operations Research,* Volume 41, pp. 31-46.

Darwin, C., 1913. *On the Origin of Species by Means of Natural Selection Or the Preservation of Favoured Races in the Struggle for Life.* s.l.:Oxford University Press.

De Souza, R. C. T., dos Santos Coelho, L., De Macedo, C. A. & Pierezan, J., 2018. A V-Shaped Binary Crow Search Algorithm for Feature Selection. *In 2018 IEEE Congress on Evolutionary Computation* , July, pp. 1-8.

Deane, J. & Agarwal, A., 2012. *Neural Metaheuristics for the Multidimensional Knapsack Problem,* s.l.: s.n.

Deane, J. & Agarwal, A., 2012. Neural, Genetic, And Neurogenetic Approaches For Solving The 0-1 Multidimensional Knapsack Problem. *International Journal of Management & Information Systems,* Volume 17, pp. 43-54.

Dell'Amico, M. & Martello, S., 1995. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing,* Volume 7, pp. 191-200.

Díaz, P. P.-C. M. et al., 2018. An improved crow search algorithm applied to energy problems. *Energies,* Volume 11, p. 571.

Dorigo, M., Maniezzo, V. & Colorni, A., 1996. Ant system: optimization by a colony of cooperating agents.. *IEEE Transactions on Systems, man, and cybernetics, Part B: Cybernetics,* Volume 26, pp. 29-41.

Drake, J. H., Hyde, M., Ibrahim, K. & Ozcan, E., 2014. A genetic programming hy-per-heuristic for the multidimensional knapsack problem. *Kybernetes,* Volume 43, pp. 1500-1511.

Drake, J., Özcan, E. & Burke, E., 2016. A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem. *Evolutionary Computation,* Volume 24, pp. 113-141.

Drexl, A., 1988. A simulated annealing approach to the multiconstraint zero-one knapsack problem.. *Computing,* Volume 40, pp. 1-8.

Driss, I., Mouss, K. & Laggoun, A., 2015. *An Effective Genetic Algorithm for the Flexible Job Shop Scheduling Problems.* Ville de Québec, s.n.

Dyckhoff, H., 1990. A typology of cutting and packing problems. *European Journal of Operational Research,* Volume 44, pp. 145-159.

Eberhart, R. & Kennedy, J., 1995. *A new optimizer using particle swarm theory. In MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, October, pp. 39-43.

Everett, H., 1963. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations research,* Volume 11, pp. 399-417.

Falkenauer, E. & Delchambre, A., 1992. A genetic algorithm for bin packing and line balancing. *In Proceedings IEEE International Conference on Robotics and Automation*, May, pp. 1186-1192.

Fekete, S. P. & Schepers, J., 2000. *On more-dimensional packing II: Bounds.*, s.l.: s.n.

Fekete, S. P. & Schepers, J., 2004. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research,* Volume 29, pp. 353-368.

Feng, X., Lau, F. C. & Gao, D., 2009. A new bio-inspired approach to the traveling salesman problem. *In International Conference on Complex Sciences*, February, pp. 1310-1321.

Fidanova, S., 2002. ACO algorithm for MKP using various heuristic information. *In International Conference on Numerical Methods and Applications* , August, pp. 438-444.

Fingler, H., Cáceres, E. N., Mongelli, H. & Song, S. W., 2014. A CUDA based solution to the multidimensional knapsack problem using the ant colony optimization. *Procedia Computer Science,* pp. 84-94.

Fleszar, K. & Hindi, K. S., 2009. Fast, effective heuristics for the 0-1 multi-dimensional knapsack problem. *Computers & Operations Research,* Volume 36, pp. 1602-1607.

Fogel, L. J., owens, A. J. & Walsh, M., 1966. *Artificial Intelligence through Simulated Evolution.* New York.: Wiley.

Frenk, J. G. & Galambos, G., 1987. Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. *Computing,* Volume 39, pp. 201-217.

Freville, A. & Plateau, G., 1994. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics,* Volume 49, pp. 189-212.

Fréville, A. & Plateau, G., 1996. The 0-1 bidimensional knapsack problem: Toward an efficient high-level primitive tool. *Journal of Heuristics,* Volume 2, pp. 147-167.

Friedman, M., 1940. Comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics,* Volume 11, p. 86–92.

Gallardo, J. E., Cotta, C. & Fernández, A. J., 2005. Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. *In International Work-Conference on the Interplay Between Natural and Artificial Computation*, june, pp. 21-30.

Gandomi, A. H., Yang, X. S. & Alavi, A. H., 2013. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems.. *Engineering with computers,* Volume 29, pp. 17-35.

Gavish, B. & Pirkul, H., 1985. Efficient algorithms for solving zero–one multidimensional knapsack problems to optimality. *Mathematical Programming,* Volume 31, pp. 78-105.

Gherboudj, A., Labed, S. & Chikhi, S., 2012. A new hybrid binary particle swarm optimization algorithm for multidimensional knapsack problem. *In Advances in Computer Science, Engineering & Applications,* pp. 489-498.

Gilmore, P. C. & Gomory, R. E., 1961. A linear programming approach to the cutting-stock problem. *Operations research,* Volume 9, pp. 849-859.

Gilmore, P. C. & Gomory, R. E., 1963. A linear programming approach to the cutting stock problem-Part II. *Operations Research,* Volume 11, pp. 863-888.

Gilmore, P. C. & Gomory, R. E., 1965. Multistage cutting stock problems of two and more dimensions. *Operations research,* Volume 13, pp. 94-120.

Gilmore, P. & Gomory, R., 1966. The theory and computation of knapsack functions. *Operations Research,* Volume 14, pp. 1045-1075.

Glover, F., 1990. Tabu search: A tutorial. *Interfaces,* Volume 20, pp. 74-94.

Glover, F. & Kochenberger, G., 1996. *Critical Event Tabu Search for Multidimensional Knapsack Problems. Meta-heuristics: Theory and Applications.* Boston: Springer.

Goel, S., Sharma, A. & Panchal, V. K., 2014. Performance analysis of bio-inspired techniques. *In Proceedings of the Third International Conference on Soft Computing for Problem Solving*, pp. 831-844.

Goh, K., Lim, A. & Rodrigues, B., 2003. Sexual selection for genetic algorithms. *Artificial Intelligence Review,* Volume 19, pp. 123-152.

Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization and Machine learning.* s.l.:Addison-Wesley.

Goldberg, D. & Lingle, R., 1985. Alleles, loci, and the travelling salesman problem. *In the First Interna-tional Conference on Genetic Algorithms and their Applications*, p. 154–159.

Gonçalves, J. F. & Resende, M., 2013. A biased random key genetic algorithm for 2D and 3D bin-packing problems. *International Journal of Production Economics,* Volume 145, pp. 500-510.

Gong, Q., Zhou, Y. & Luo, Q., 2011. Hybrid Artificial Glowworm Swarm Optimization Algorithm for Solving Multidimensional Knapsack Problem. *Procedia Engineering,* Volume 15, pp. 2880-2884..

Greenberg, H. J. & Pierskalla, W. P., 1970. Surrogate mathematical programming. *Operations Research,* Volume 18, pp. 924-939.

Green, C. J., 1967. *Two Algorithms for Solving the Independent Multi-dimensional Knapsack Problems,* s.l.: s.n.

Gupta, I. K., Choubey, A. & Choubey, S., 2017. Clustered genetic algorithm to solve multidimensional knapsack problem. *In 8th International Conference on Computing, Communication and Networking Technologies*, july, pp. 1-6.

Gupta, S. & Arora, S., 2015. Boosting simulated annealing with fitness landscape parameters for better optimality. *International Journal of Computing,* Volume 14, pp. 107-112.

Hamdi-Dhaoui, K., Labadie, N. & Yalaoui, A., 2012. Algorithms for the two dimensional bin pack-ing problem with partial conflicts. *RAIRO-Operations Research,* Volume 46, pp. 41-62.

Hanafi, S., Lazić, J., Mladenović, N. & Wilbaut, C., 2009. Variable Neighbourhood Decomposition Search with Bounding for Multidimensional Knapsack Problem. *IFAC Proceedings Volumes,* Volume 42, pp. 2018-2022.

Hanafi, S. & Wilbaut, C., 2008. Scatter Search for the 0-1 Multidimensional Knapsack Problem. *Journal of Mathematical Modelling and Algorithms,* Volume 7, pp. 143-159.

Hanafi, S. & Wilbaut, C., 2011. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals of Operations Research,* Volume 183, pp. 125-142.

Hansen, P. & Mladenović, N., 1997. Variable neighborhood search. *Computers and operations research,* Volume 24, pp. 1097-1100.

Hoff, A., Løkketangen, A. & Mittet, I., 1996. Genetic algorithms for 0/1 multidimensional knapsack problems. *In Proceedings Norsk Informatikk Konferanse*, November, pp. 291-301.

Holland, J., 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Hopper, E. & Turton, B., 1999. A genetic algorithm for a 2D industrial packing problem. *Computers & Industrial Engineering,* Volume 37, pp. 375-378.

Huang, K. W. & Wu, Z. X., 2018. CPO: A Crow Particle Optimization Algorithm. *International Journal of Computational Intelligence Systems,* Volume 12, pp. 426-435.

Jain, A. K., N., M. M. & Flynn, P. J., 1999. Data clustering: a review. *ACM computing surveys* , pp. 264-323.

Jain, M., Rani, A. & Singh, V., 2017. An improved Crow Search Algorithm for high-dimensional problems. *Journal of Intelligent & Fuzzy Systems,*, Volume 33, pp. 3597-3614.

James, R. J. & Nakagawa, Y., 2005. Enumeration methods for repeatedly solving multidimensional knapsack sub-problems. *IEICE TRANSACTIONS on Information and Systems,* Volume 88, pp. 2329-2340.

Jeong, Y. W., Park, J. B., Jang, S. H. & Lee, K. Y., 2010. A new quantum-inspired binary PSO: application to unit commitment problems for power systems. *IEEE Transactions on Power Systems,* Volume 25, pp. 1486-1495.

Ji, J., Wei, H., Liu, C. & Yin, B., 2013. Artificial Bee Colony Algorithm Merged with Pheromone Communication Mechanism for the 0-1 Multidimensional Knapsack. *Mathematical Problems in Engineering,* Volume 13.

Johnson, D. S. & Garey, M. R., 1979. *Computers and intractability: A guide to the theory of NP-completeness.* San Francisco: s.n.

Karaboga, D., 2005. *An idea based on honey bee swarm for numerical optimization,* s.l.: s.n.

Karp, R. M., 1972. Complexity of computer computations. Dans: *Reducibility among combinatorial problems.* Boston, MA: Springer, pp. 85-103.

Ke, L., Feng, Z., Ren, Z. & Wei, X., 2010. An ant colony optimization approach for the multidimensional knapsack problem. *Journal of Heuristics,* Volume 16, pp. 65-83.

Kellerer, H., Pferschy, U. & Pisinger, D., 2004. *Knapsack problems.* New York: Springer-Verlag Berlin Heidelberg.

Kenmochi, M. et al., 2009. Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research,* Volume 198, pp. 73-83.

Kennedy, J. & Eberhart, R., 1995. Particle swarm optimization (PSO). *In Proceeding of IEEE International Conference on Neural Networks, Australia* , November, pp. 1942-1948.

Kennedy, J. & Eberhart, R. C., 1997. A discrete binary version of the particle swarm algorithm.. *In 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*, October, Volume 5, pp. 4104-4108.

Khalili-Damghani, K. & Taghavifard, M., 2011. Solving a bi-objective project capital budgeting problem using a fuzzy multi-dimensional knapsack. *Journal of Industrial En-gineering International,* pp. 67-73.

Khanafer, A., Clautiaux, F. & Talbi, E. G., 2012. Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts.. *Computers & Operations Research,* Volume 39, pp. 54-63.

Khuri, S., Bäck, T. & Heitkötter, J., 1994. The Zero/One Multiple Knapsack Problem and Genetic Algorithms. *Proceedings of the ACM Symposium on Applied Computing*, pp. 188-193.

Király, A. & Abonyi, J., 2010. A novel approach to solve multiple traveling salesmen problem by genetic algorithm. *In Computational Intelligence in Engineering,* pp. 141-151.

Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P., 1983. Optimization by simulated annealing. *science,* Volume 220, pp. 671-680.

Klir, G. J. & Yuan, B., 1995. *Fuzzy sets and fuzzy logic.* New Jersey: Prentice Hall PTR.

Kochenberger, G., McCarl, B. & Wymann., F., 1974. A heuristic for general integer programming. *Decision Sciences,* Volume 5, pp. 36-44.

Kong, M., Tian, P. & Kao, Y., 2008. A new ant colony optimization algorithm for the multidimensional knapsack problem. *Computers & Operations Research,* Volume 35, pp. 2672-2683.

Kong, X., Gao, L., Ouyang, H. & Li, S., 2015. Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm. *Computers & Operations Research,* Volume 63, pp. 7-22.

Kong, X., Gao, L., Ouyang, H. & Li, S., 2015. Solving Large-Scale Multidimensional Knapsack Problems with a New Binary Harmony Search Algorithm. *Computers & Operations Research,* Volume 63, pp. 7-22.

Kunikazu, Y. & Prékopa, A., 2012. *Convexity and Solutions of Stochastic Multidimensional Knapsack Problems with Probabilistic Constraints,* s.l.: s.n.

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B., 2018a. The 0/1 Multidimensional Knapsack Problem and Its Variants: A Survey of Practical Models and Heuristic Approaches. *American Journal of Operations Research,* Volume 8, pp. 395-439. https://doi.org/10.4236/ajor.2018.85023.

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B., 2018b. A hybrid genetic algorithm for solving 0/1 Knapsack Problem. *In Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications*, 2-5 May. Rabat/ Morocco, p. 6 pages. DOI: https://doi.org/10.1145/3230905.3230907..

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B., 2018c. A genetic algorithm based on k-means method for solving 0/1 multidimensional knapsack problem. *Presented in the International Conference of Computer Science and Renewable Energies*, 22-23 November. Ouarzazat/Morocco.

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B., 2019a. An improved sexual genetic algorithm for solving 0/1 multidimensional knapsack problem. *Engineering Computations,* 36(7), pp. 2260-2292. https://doi.org/10.1108/EC-01-2019-0021.

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B., 2019b. A crow search based genetic algorithm for solving two dimensional bin packing problem. *In: Benzmüller C., Stuckenschmidt H. (eds) KI 2019: Advances in Artificial Intelligence. KI 2019 (Kassel-Germany). Lecture Notes in Computer Science*, 23-26 September, Volume 11793, pp. 203-215. https://doi.org/10.1007/978-3-030-30179-8_17.

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B., 2020. A binary crow search algorithm for solving two dimensional bin packing problem with fixed orientation. *Presented in the International Conference on Computational Intelligence (Gurugram-India). Procedia Computer Science,* 6-7 September, Volume 167, pp. 809-818. https://doi.org/10.1016/j.procs.2020.03.420.

Lai, G., Yuan, D. & Yang, S., 2014. A new hybrid combinatorial genetic algorithm for multidimensional knapsack problems. *Journal of Supercomputing,* Volume 70, pp. 930-945.

Lakshmi, K., Visalakshi, N. K. & Shanthi, S., 2018. Data clustering using K-Means based on Crow Search Algorithm. *Sādhanā,* Volume 43, p. 190.

Langeveld, J. & Engelbrecht, A. P., 2012. Set-based particle swarm optimization applied to the multidimensional knapsack problem. *Swarm Intelligence,* Volume 6, pp. 297-342.

Lee, J. S. & Guignard, M., 1988. An Approximate Algorithm for Multidimensional Zero-One Knapsack Problems-A Parametric Approach. *Management Science,* Volume 34, pp. 402-410.

Leguizamon, G. & Michalewicz, Z., 1999. A new version of ant system for subset problems. *In IEEE Proceedings of the 1999 Congress on Evolutionary Computation*, Volume 2, pp. 1459-1464.

Leung, S. C., Zhang, D., Zhou, C. & Wu, T., 2012. A hybrid simulated annealing me-taheuristic algorithm for the two-dimensional knapsack packing problem. *Computers & Operations Research,* Volume 39, pp. 64-73.

Leung, S. C., Zhou, X., Zhang, D. & Zheng, J., 2011. Extended guided tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem. *Computers & Operations Research*, Volume 38, pp. 205-215.

Liang, Y. C. & Cuevas Juarez, J. R., 2016. A novel metaheuristic for continuous optimization problems: Virus optimization algorithm. *Engineering Optimization*, Volume 48, pp. 73-93.

Likas, A., Vlassis, N. & Verbeek, J. J., 2003. The global k-means clustering algorithm.. *Pattern recognition,* Volume 36, pp. 451-461.

Lin, C. J., Chern, M. S. & Chih, M., 2016. A binary particle swarm optimization based on the surrogate information with proportional acceleration coefficients for the 0-1 multidimensional knapsack problem.. *Journal of Industrial and Production Engineering,* Volume 33, pp. 77-102.

Lin, G. & Yao, X., 1997. Analysing crossover operators by search step size. *In Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 107-110.

Lis, J. & Eiben, Á., 1996. A multi-sexual genetic algorithm for multiobjective optimization. *In Proceedings of the IEEE International Conference on Evolutionary Computation, Japan*, pp. 59-64.

Liu, D. S., Tan, K. C., Goh, C. K. & Ho, W. K., 2006. On solving multiobjective bin packing problems using particle swarm optimization. *In IEEE International Conference on Evolutionary Computation*, July , pp. 2095-2102.

Liu, D. S. et al., 2008. On solving multiobjective bin packing problems using evolutionary particle swarm optimization.. *European Journal of Operational Research,* Volume 190, pp. 357-382.

Liu, J. et al., 2016. A Binary Differential Search Algorithm for the 0-1 Multidimensional Knapsack Problem. *Applied Mathematical Modelling,* Volume 40, pp. 9788-9805.

Liu, R. T. & Lv, X. J., 2013. MapReduce-based ant colony optimization algorithm for multi-dimensional knapsack problem. *In Applied Mechanics and MaterialsTrans Tech Publications,* Volume 380, pp. 1877-1880.

Lodi, A., Martello, S. & Vigo, D., 1999. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing,* p. 345–357.

Lodi, A., Martello, S. & Vigo, D., 2002. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics,* Volume 123, pp. 379-396.

Løkketangen, A. & Glover, F., 1996. *Probabilistic Move Selection in Tabu Search for Zero-One Mixed Integer Programming Problems Programming Problems. Meta-Heuristics: Theory and Applications.* s.l.:Kluwer Academic Publishers.

Lorie, J. H. & Savage, L. J., 1955. Three problems in rationing capital. The Journal of Business, 28. *The Journal of Business,* pp. 229-239..

Loulou, R. & Michaelides, E., 1979. New greedy-like heuristics for the multidimensional 0-1 knapsack problem. *Operations Research,* Volume 27, pp. 1101-1114.

MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. *In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, June, Volume 1, pp. 281-297.

Magazine, M. J. & Oguz, O., 1984. A heuristic algorithm for the multidimensional zero-one knapsack problem. *European Journal of Operational Research,* Volume 16, pp. 319-326.

Mann, A. K. & Kaur, N., 2013. Survey paper on clustering techniques. *International Journal of Science, Engineering and Technology Research,* Volume 2, pp. 803-806.

Mansini, R. & Speranza, M. G., 2012. Coral: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing,* Volume 24, pp. 399-415.

Marichelvam, M. K. & Geetha, M., 2018. A hybrid crow search algorithm to minimise the weighted sum of makespan and total flow time in a flow shop environment.. *International Journal of Computer Aided Engineering and Technology,* Volume 10, pp. 636-649.

Martello, S., Pisinger, D. & Vigo, D., 2000. The three-dimensional bin packing problem. *Operations research,* Volume 48, pp. 256-267.

Martello, S. & Toth, P., 1990. *Knapsack Problems: Algorithms and Computer Implementations.* New York: John Wiley.

Martello, S. & Vigo, D., 1998. Exact solution of the two-dimensional finite bin packing problem. *Management science,* Volume 44, pp. 388-399.

Martins, J. P., Fonseca, C. M. & Delbem, A. C., 2014. On the performance of linkage-tree genetic algorithms for the multidimensional knapsack problem. *Neurocomputing,* Volume 146, pp. 17-29.

Meng, T., Duan, J. & Pan, Q., 2017. An Improved Migrating Birds Optimization. *Proceedings of the 29th IEEE Chinese Control and Decision Conference*, 28-30 May, pp. 4698-4703.

Metropolis, N., Rosenbluth, M. N. & Teller, H. A., 1953. Equation of state calculation by fast computing machines. *Journal of Chemical Physics,* Volume 21, pp. 1087-1092.

Miller, G., 1994. Exploiting mate choice in evolutionary computation: Sexual selection as a process of search, optimization, and diversification. *Lecture Notes in Computer Science.*

Miller, G. & Todd, P., 1995. The role of mate choice in biocomputation: Sexual selection as a process of search, optimization, and diversification. *Lecture Notes in Computer Science.*

Mingwei, L. I. et al., 2014. Binary glowworm swarm optimization for unit commitment. *Journal of Modern Power Systems and Clean Energy,* Volume 2, pp. 357-365.

Mirjalili, S., 2016. SCA: a sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems,* Volume 96, pp. 120-133.

Mirjalili, S. & Lewis, A., 2013. S-shaped versus V-shaped transfer functions for binary particle swarm optimization. *Swarm and Evolutionary Computation,* Volume 9, pp. 1-14.

Mirjalili, S., Mirjalili, S. M. & Lewis, A., 2014. Grey wolf optimizer. *Advances in engineering software,* Volume 69, pp. 46-61.

Mohammadi, F. & Abdi, H., 2018. A modified crow search algorithm (MCSA) for solving economic load dispatch problem. *Applied Soft Computing,* Volume 71, pp. 51-65.

Monaci, M. & Toth, P., 2006. A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing,* Volume 18, pp. 71-85.

Montaña, J. L., Alonso, C. L., Cagnoni, S. & Callau, M., 2008. Computing surrogate constraints for multidimensional knapsack problems using evolution strategies. *Proceedings of the Workshops on Applications of Evolutionary Computation. Lecture Notes in Computer Sciences*, Volume 4974, pp. 555-564.

Nakbi, W., Alaya, I. & Zouari, W., 2015. A hybrid lagrangian search ant colony optimization algorithm for the multidimensional knapsack problem. *Procedia Computer Science,* Volume 60, pp. 1109-1119.

Neri, F. & Tirronen, V., 2010. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review,* Volume 33, pp. 61-106.

Niar, S. & Freville, A., 1997. A parallel tabu search algorithm for the 0-1 multidimensional knapsack problem. *In IEEE Proceedings 11th International Parallel Processing Symposium*, pp. 512-516.

Parreño, F., Alvarez-Valdés, R., Oliveira, J. F. & Tamarit, J. M., 2010. A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing. *Annals of Operations Research,* Volume 179, p. 203–220.

Pasandideh, S. H. R. & Khalilpourazari, S., 2018. *Sine cosine crow search algorithm: a powerful hybrid meta heuristic for global optimization,* s.l.: s.n.

Peng, H., Wu, Z., Shao, P. & Deng, C., 2016. Dichotomous binary differential evolution for knapsack problems. *Mathematical Problems in Engineering.*

Pfeiffer, J. & Rothlauf, F., 2007. Analysis of greedy heuristics and weight-coded EAs for multidimensional knapsack problems and multi-unit combinatorial auctions. *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 1529-1529.

Pham, D. T. et al., 2005. *The bees algorithm,* UK.: s.n.

Pirkul, H., 1987. A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem. *Naval Research Logistics,* Volume 34, pp. 161-172.

Pisinger, D. & Sigurd, M., 2007. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, Volume 19, pp. 36-51.

Puchinger, J. & Raidl, G., 2005. *Relaxation guided variable neighborhood search.* s.l., s.n.

Puchinger, J. & Raidl, G. R., 2004. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In International Conference on Parallel Problem Solving from Nature. *In International Conference on Parallel Problem Solving from Nature*, September, pp. 642-651.

Puchinger, J. & Raidl, G. R., 2008. Bringing order into the neighborhoods: relaxation guided variable neighborhood search. *Journal of Heuristics*, Volume 14, pp. 457-472.

Puchinger, J., Raidl, G. R. & Pferschy, U., 2006. The core concept for the multidimensional knapsack problem. *In European Conference on Evolutionary Computation in Combinatorial Optimization*, April, pp. 195-208.

Raidl, G. R., 1998. An improved genetic algorithm for the multiconstrained 0-1 knap-sack problem. *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 207-211.

Raidl, G. R., 1999. Weight-codings in a genetic algorithm for the multi-constraint knapsack problem. *In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, july, Volume 1, pp. 596-603.

Rechenberg, I., 1965. Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Library Translation,* Volume 1122.

Rezoug, A., Boughaci, D. & Badr-El-Den, M., 2015. Memetic algorithm for solving the 0-1 multidimensional knapsack problem. *In Portuguese Conference on Artificial Intelligence*, September, pp. 298-304.

Sabba, S. & Chikhi, S., 2014. A Discrete Binary Version of Bat Algorithm for Multidimensional Knapsack Problem. *International Journal of Bio-Inspired Computation,* Volume 6, pp. 140-152.

Sandholm, T., Suri, S., Gilpin, A. & Levine, D., 2002. Winner determination in combinatorial auction generalizations. *roceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. ACM*, May, pp. 69-76.

Sarin, S. C. & Wilhelm, W. E., 1984. Prototype models for two-dimensional layout design of robot systems. *IIE transactions,* Volume 16, pp. 206-215.

Scheithauer, G., 1991. A three-dimensional bin packing algorithm. *Elektronische Informationsverarbeitung und Kybernetik,* Volume 27, pp. 263-271.

Schulze, B., 2017. *New perspectives on multi-objective knapsack problems,* Germany: University of Wuppertal.

Scicon, L., 1986. *SCICONIC/VM users guide.* s.l.:s.n.

Senju, S. & Toyoda, Y., 1968. An approach to linear programming with 0-1 variables. *Management Science,* Volume 15, pp. 196-207.

Sheta, A. F., 2017. Solving the Economic Load Dispatch Problem Using Crow Search Algorithm. *In 8th International Multi-Conference on Complexity, Informatics and Cybernetics* , pp. 95-100.

Shih, W., 1979. A branch and bound method for the multiconstraint knapsack problem. *Journal of the Operational Research Society,* Volume 30, p. 369–378.

Shin, Y. B. & Kita, E., 2017. Solving two-dimensional packing problem using particle swarm optimization.. *Computer Assisted Methods in Engineering and Science,* Volume 19, pp. 241-255.

Sierry & Collette, 2002. *optimisation Muliobjective.* s.l.:s.n.

Soke, A. & Bingul, Z., 2006. Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. *Engineering Applications of Artificial Intelligence,* Volume 19, pp. 557-567.

Sollow, 2007. Linear and nonlinear programming. *Wiley Encyclopedia of Computer Science and Engineering.*

Song, Y., Zhang, C. & Fang, Y., 2008. Multiple Multidimensional Knapsack Problem and Its Applications in Cognitive Radio Networks. *Proceedings of the IEEE Military Communications Conference (MILCOM 2008)*, pp. 1-7.

Soyster, A. L., Lev, B. & Slivka, W., 1978. Zero-one programming with many variables and few constraints. *European Journal of Operational Research,* Volume 2, pp. 195-201.

Spieksma, F. C., 1994. A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers & operations research,* Volume 21, pp. 19-25.

Stille, W., 2008. *Solution techniques for specific bin packing problems with applications to assembly line optimization,* s.l.: s.n.

Taillandier, F., Fernandez, C. & Ndiaye, A., 2017. Real Estate Property Maintenance Optimization Based on Multiobjective Multidimensional. *Computer-Aided Civil and Infrastructure Engineering,* p. 227–251.

Tasgetiren, M. F., Pan, Q. K., Kizilay, D. & Suer, G., 2015. A differential evolution algorithm with variable neighborhood search for multidimensional knapsack problem. *In 2015 IEEE Congress on Evolutionary Computation (CEC)*, May, pp. 2797-2804.

Thesen, A., 1975. A recursive branch and bound algorithm for the multidimensional knapsack problem. *Naval Research Quarterly,* Volume 22, p. 341–353.

Thiel, J. & Voss, S., 1994. Some experiences on solving multiconstraint zero-one knapsack problems with genetic algorithms. *INFOR: Information Systems and Operational Research,* Volume 32, pp. 226-242.

Toyoda, Y., 1975. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science,* Volume 21, pp. 1417-1427.

Turgut, M. S. & Turgut, O. E., 2017. Hybrid Artificial Cooperative Search–Crow Search Algorithm for Optimization of a Counter Flow Wet Cooling Tower. *International Journal of Intelligent Systems and Applications in Engineering,* Volume 5, pp. 105-116.

Uymaz, S. A., Tezel, G. & Yel, E., 2015. Artificial algae algorithm (AAA) for nonlinear global optimization. *Applied Soft Computing,* Volume 31, pp. 153-171.

Varnamkhasti, J. M. & Lee, L. S., 2012b. A Fuzzy Genetic Algorithm Based on Binary Encoding for Solving Multidimensional Knapsack Problems. *Journal of Applied Mathematics.*

Varnamkhasti, M. J. & Lee, L. S., 2012a. A genetic algorithm based on sexual selection for the multidimensional 0/1 knapsack problems. *International Journal of Modern Physics: Conference Series.,* Volume 9, pp. 422-431.

Vasquez, M. & Hao., J. K., 2001. An hybrid approach for the 0-1 multidimensional knapsack problem. *Proc. 17th Internat. Joint Conf.*, pp. 328-333.

Vasquez, M. & Hao, J.-K., 2001. Une approche hybride pour le sac à dos multidimensionnel en variables 0-1. *RAIRO-Operations Research,* Volume 35, p. 415–438.

Vasquez, M. & Vimont, Y., 2005. Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research,* Volume 165, pp. 70-81.

Vimont, Y., Boussier, S. & Vasquez., M., 2008. Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. *J. Combin. Optim.,* Volume 15, pp. 165-178.

Volgenant, A. & Zoon, J., 1990. An Improved Heuristic for Multidimensional 0-1 Knapsack Problems. *Journal of the Operational Research Society,* Volume 41, pp. 963-970.

Volgenant, A. & Zwiers, I., 2007. Partial Enumeration in Heuristics for Some Combinatorial Optimization Problems. *Journal of Operational Research Society,* Volume 58, pp. 73-79.

Wang, L., Wang, X. T. & Fei, M. R., 2009. An adaptive mutation-dissipation binary particle swarm optimisation for multidimensional knapsack problem. *International Journal of Modelling, Identification and Control,* Volume 8, pp. 259-269.

Wang, L. et al., 2015. A Human Learning Optimization Algorithm and Its Application to Multi-Dimensional Knapsack Problems. *Applied Soft Computing*, Volume 34, pp. 736-743..

Wang, L., Zheng, X. L. & Wang, S. Y., 2013. A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Knowledge-Based Systems*, Volume 48, pp. 17-23.

Wang, W. B., Feng, Q. & Liu, D., 2012. Synthesis of thinned linear and planar antenna arrays using binary PSO algorithm. *Progress In Electromagnetics Research*, Volume 127, pp. 371-387.

Wäscher, G., Haußner, H. & Schumann, H., 2007. An improved typology of cutting and packing problems. *European journal of operational research*, Volume 183, pp. 1109-1130.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. *Biometrics*, Volume 1, pp. 80-83.

Wong, L., 2009. *Heuristic Placement Routines For Two-Dimensional Rectangular Bin Packing Problems.*, s.l.: s.n.

Xu, R. & Wunsch., D. C., 2005. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, Volume 16, pp. 645 - 678.

Yang, H., Wang, M. & Yang, C., 2013. A hybrid of rough sets and genetic algorithms for solving the 0-1 multidimensional knapsack problem. *International Journal of Inno-vative Computing, Information and Control*, Volume 9, pp. 3537-3548.

Yang, X. S., 2010. A new metaheuristic bat-inspired algorithm.. *In Nature inspired cooperative strategies for optimization*, pp. 65-74.

Yang, X. S., 2012. Flower pollination algorithm for global optimization.. *In International conference on unconventional computing and natural computation*, September, pp. 240-249.

Ykman-Couvreur, C., Nollet, V., Catthoor, F. & Corporaal, H., 2001. Fast multidimension multichoice knapsack heuristic for MP-SoC runtime management. *ACM Trans. Embedd. Comput. Syst.*, April, p. 16.

Yoon, Y. & Kim, Y. H., 2013. A memetic Lagrangian heuristic for the 0-1 multidi-mensional knapsack problem. *Discrete Dynamics in Nature and Society.*

Yoon, Y., Kim, Y. H. & Moon, B. R., 2012. A theoretical and empirical investigation on the Lagrangian capacities of the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, Volume 218, pp. 366-376.

Yuan, Q. & Yang, Z., 2016. A Weight-Coded Evolutionary Algorithm for the Multidimensional Knapsack Problem. *Advances in Pure Mathematics*, Volume 6, pp. 659-675.

Zhang, Q., Xie, Q. & Wang, G., 2016. A survey on rough set theory and its applications. *CAAI Transactions on Intelligence Technology*, Volume 1, pp. 323-333.

Zhang, X. et al., 2016. Artificial Algae Algorithm for Multidimensional Knapsack Problems. *Applied Soft Computing*, Volume 43, pp. 583-595..

Zhang, X. et al., 2016. Binary artificial algae algorithm for multidimensional knapsack problems. *Applied Soft Computing*, Volume 43, pp. 583-595.

Zouache, D., Nouioua, F. & Moussaoui, A., 2016. Quantum-inspired firefly algorithm with particle swarm optimization for discrete optimization problems. *Soft Computing*, Volume 20, pp. 2781-2799.

N° d'ordre: CED-SFA/05-20

Résumé:

La présente thèse porte sur les approches heuristiques pour résoudre les problèmes de chargement. Nous avons focalisé notre attention sur deux types de problèmes : Un problème de maximisation et un problème de minimisation. Pour le premier type, on a pris comme exemple le fameux problème de sac à dos multidimensionnel binaire (0/1 MKP). Tandis que pour le deuxième type, on a opté pour le problème de bin packing à deux dimensions (2D-BPP). Ces deux problèmes ont suscité l'attention de plusieurs chercheurs au regard d'autres problèmes de chargement et continuent d'être enrichis par une bibliographie assez étendue. Sur le plan pratique, ces deux problèmes connaissent un regain d'intérêt majeur, notamment sur le secteur industriel et économique. Ces deux problèmes ont modélisé maints problèmes réels, tel que le problème d'affectation des ressources, le problème d'ordonnancement des tâches, le problème de découpe, comme ils font l'objet d'un grand intérêt dans le transport et la logistique.

Les deux problèmes appartiennent aux problèmes NP-difficiles de l'optimisation combinatoire. En d'autres mots, il n'existe pas un algorithme exact permettant de trouver une solution optimale en un temps polynomial. Cependant, nous pouvons utiliser des méthodes approchées, notamment les heuristiques et les métaheuristiques, pour trouver des solutions de bonne qualité en un temps raisonnable mais sans garantir l'optimalité des solutions. Nous proposons dans cette thèse diverses métaheuristiques pour résoudre les deux problèmes de chargement. Nous développons un algorithme génétique amélioré pour résoudre le 0/1 MKP, ainsi qu'une approche hybride qui incorpore une méthode de classification des k-moyennes dans un algorithme génétique. De plus, nous concevons une variante d'une métaheuristique récente basée sur l'intelligence en essaim, à savoir l'algorithme de recherche des corbeaux. Ce dernier algorithme est binarisé afin qu'il soit adapté au contexte du problème du 2D-BPP, en appliquant deux techniques différentes: Une technique de binarisation et une technique d'hybridation. Ces méthodes proposées ont été testées expérimentalement sur des jeux d'essai existant dans la littérature des deux problèmes. Leurs résultats démontrent clairement leur efficacité en termes de qualité des solutions et du temps du calcul. Finalement, on note que les métaheuristiques proposées peuvent être appliquées aux divers problèmes d'optimisation pour le calcul de solutions approchées.

Mots clés

Sac à dos multidimensionnel, Bin packing en deux dimensions, Optimisation, Métaheuristiques, Algorithme génétique, Algorithme de recherche des corbeaux, Méthode des *k*-moyennes, Hybridation, Binarisation.