Effective Solving of The 0-1 Multidimensional Knapsack Problem

By

ABDELLAH REZOUG

Supervised by

DALILA BOUGHACI

جامعة هواري بومدين
للعلوم والتكنولوجيا
USTHB

Department of Computer Science
UNIVERSITY OF SCIENCES AND TECHNOLOGY HOUARI BOUMEDIENE (USTHB)

Thesis submitted to the USTHB in accordance with the requirements of the degree of DOCTOR OF SCIENCE in the Faculty of Computer Science and Electronics.

Dedication and acknowledgements

This thesis is dedicated to my parents for their love and support throughout my life. Mother, thank you for the unconditional endless love. Father, you always teach me how to be a good man, thank you for your support. May God keep you among us for a long long time. My sisters and brothers deserve my wholehearted thanks. Your presence has always given me strength to believe in my dreams.

I would like to express my gratitude to Prof. Dalila Boughaci, professor at the University of Sciences and Technology Houarie Boumediene, for supervising my work. I thank her for her availability, support and advice throughout the years of preparation of my thesis project.

I would also like to sincerely thank Dr. Mohamed Bader-El-Den. His hospitality, ideas, encouragement have supported, enlightened, and entertained me over the year of my scholarship at Portsmouth, UK.

The members of my thesis committee, have generously given their time and expertise to better my work. I thank them for their contribution and their good-natured support.

I need to express my gratitude and deep appreciation to all my friends and colleagues.

Abstract

This thesis sights to propose hybrid heuristic methods that effectively address the 0-1 Multidimensional Knapsack Problem (MKP01). MKP01 is a well-known NP-hard combinatorial optimisation problem that has received a wide attention from researchers relative to its multiple applications such as project selection, resource allocation, etc. In the last decades, a huge number of exact, approximate and hybrid methods have been suggested. This work will focus on approximate and hybrid solutions.

With a will to cover different types of solutions, this study attempts to investigate heuristics from simple to complex, starting by the local search algorithms, then hybrid population-based and local search algorithms and finishing by hybrid knowledge and population-based heuristics; therefore, the work is divided into three parts accordingly. Firstly, Simulated Annealing (SA) and Stochastic Local Search are unified to produce a new local search algorithm named SLSA. Additionally, SA, SLS, Tabu Search (TS) and Hill-Climbing (HC) algorithms are applied to the Winner Determination Problem in Multi-Unit Combinatorial Auctions (MU-WDP) which is an application of the MKP01.

Secondly, Genetic Algorithm is combined with SA and with SLS, then, is combined with SLSA. Also, Harmony Search (HS) is improved to the Self-Adaptive Harmony Search (SAHS) by adding new strategies to measure the Pitch Adjusting Rule (PAR) and the Bandwidth (bw); SAHS is also hybridised with SLS.

Thirdly, a two-step heuristic that consists of a greedy algorithm which extracts a useful acknowledge about the structure of the optimal solutions combined with an acknowledge-guided GA. The proposed approach, denoted Guided Genetic Algorithm (GGA), first performs a pre-analysis of the data using greedy algorithm and then, utilises the collected information to guide the process of a standard GA.

Several experimental studies were conducted to evaluate the performance of the proposed approaches on well-known MKP01 data. Three sets of data have been used: a set of simple and small-size data, two sets of large and complex data. Despite some benchmarks may probably be considered as old, the optimal solution of some instances are still unknown. Further experiments were undertaken that aimed to compare our methods to the literature as well.

The results revealed that SLS, SLSA, GA-SLS, MSA, HAHS-SLS and GGA were capable to achieve high-quality solutions and were competitive with the literature. Consequently, they are most likely to contribute to solve other combinatorial optimisation problems. The finding of this work indicated that further work still could be done to increase the upper bounds of the MKP01, in particular, in the field of hybrid heuristics.

Table of Contents

List of Tables

Table **Page**

LIST OF FIGURES

Chapter 1

Introduction

Combinatorial Optimisation (CO) is one of the most emergent fields of applied mathematics and computer science. It has been extensively used in diverse domains including industry, economy, manufacturing, engineering, etc. CO is the domain of applied mathematics and of computer science which consists in finding, among a discrete set of objects, the subset of objects which represents the optimal configuration. Before becoming a full domain, CO was a branch of the Combinatorics. As a discipline of mathematical, CO is relatively young (Schrijver, 2005). The Linear and Integer Programming (LP, IP respectively) leaded to the creation of CO at the end of the forties. Moreover, problems that previously were independent, were formulated with mathematical models in CO.

Our ability to solve big and complex CO problems is improved in a spectacular way during the last decades. Methods and tools were suggested in a will of tackling all the models of CO's problems by one tool. Diverse applications in various domains were modelled as CO problems. Therefore, the 0-1 Multidimensional Knapsack Problem (MKP01) was one of the most studied CO problems during the second half of the twentieth century. The work presented in this thesis can be classified in the CO domain. Three types of solution are introduced to solve the 0-1 Multidimensional Knapsack Problem (MKP01).

MKP01 is a strong NP-hard CO problem (Hartmanis, 1982). This problem is a generalised version of 0-1 Knapsack Problem (KP01). Many names have been used such as: Multi-knapsack problem, m-dimensional, knapsack problem, multi-constraints knapsack Problem 0-1, etc.

MKP01 has been extensively considered owing to its theoretical importance for a wide range of applications. Many practical engineering design problems can be formulated as MKP01 such as: capital budgeting (Meier et al., 2001; Gilmore & Gomory, 1966), project selection (Beaujon et al., 2001), resource allocation in distributed systems (Gavish & Pirkul, 1985) and cutting stock (Shih, 1979) problems. Moreover, MKP01 can be seen as a general model for any kind of binary problems with positive coefficients(Hooker et al., 1994; Kochenberger et al., 1974).

MKP01 is composed of n items and one knapsack of m dimensions. Each item is characterised by its weight and its value. Weight of an item i is given by a m-dimensional vector $w_{ij} = \langle w_{i1}, w_{i2}, \ldots, w_{im} \rangle$ and knapsack has a m-dimensional capacity vector $\langle w_1, w_2, \ldots, w_m \rangle$. The target is to maximise the total value of items in the knapsack so that the sum of weights in each dimension j does not exceed its capacity w_j.

MKP01 has received a wide attention from the CO research community. Methods for solving MKP01 can be classified into exact, approximate and hybrid. Exact methods are used for small-size problems. Branch and Bound (B&B), Branch and Cut (B&C), Linear, Dynamic and Quadratic Programming (LP, DP, QP respectively), etc. are the principal exact methods used for solving MKP01 (Fukunaga, 2011; Yoon & Kim, 2013). The advantage of exact methods is their ability to return optimal solutions, but their processing time is exponential with the data-size of the tackled problem. Problems with small-size data are most adapted to this category.

Approximate methods are capable to give solutions close to optimal solutions. They are mainly based on heuristics such as: Simulated Annealing (SA), Tabu

Search (TS), Genetic Algorithm (GA), Ant Colony Optimisation (ACO), Particle Swarm Optimisation (PSO), Harmony Search (HS), etc. (Cho & Kim, 1997; Chu & Beasley, 1998; Vasquez & Vimont, 2005). Approximate methods have the advantage of giving good solutions within a reasonable time, consequently, they are used for solving large MKP01. The inconvenience is that these methods are uncertain to reach solutions of high quality, because they have been validated based on experimental studies.

Hybrid methods combine two or more exact or/and approximate methods. In a hybrid method, multiple methods of different kind are united in a way to improve the performance of each other. Usually, meta-heuristics are combined with others meta-heuristics or with exact methods. Hybrid methods are the most used for MKP01 such as (Chih et al., 2014; Deane & Agarwal, 2013; Della Croce & Grosso, 2012; Djannaty & Doostdar, 2008; Ke et al., 2010; Feng et al., 2014; Tuo et al., 2014) and so on. Obtained solutions are approximate but of high quality and using exact methods may guaranty this quality by calculating upper and lower bounds.

The central aim of this thesis is to develop heuristic methods that deal effectively with the MKP01. This study intends to combine (hybridise) existing heuristics and possibly, propose new versions that solve effectively large MKP01 problems as well as improve significantly the existing ones. These methods will be inspired and created based on existing methods. Therefore, this work attempts to investigate the literature review, in order to determine strength and weakness of methods in literature. Additionally, the study needs to conduct experimental tests to examine the developed methods on state-of-the-art experimental data. Furthermore, the work must validate the suggested approaches via comparing them with the best results in the literature in terms of performance, processing time, etc.

The data used to undertake experimental studies is composed of three sets. The first (usually 'SAC-94' is used to refer to this set) set consists in simple and small-size problems. The number of variables (i.e. items) n ranges from 6 to 105 and the number of constraints (i.e. dimensions) m from 2 to 30. The second set consists in large-size and complex problems ('chubeas' refers to this set). The number of variables and constraints is equal to $n = \{100, 250, 500\}$ and $m = \{5, 10, 30\}$ respectively. This set is divided into nine classes; each class contains thirty instances of MKP01. The third set consists in more complex and larger data; where n ranges from 100 up to 2500 and m ranges from 15 up to 50. Eleven instances compose this data, for most of them, the optimal solution is still unknown. Furthermore, as data for the MU-WDP is unavailable, a set of instances is generated randomly. all this data are widely used by researchers and are available online in the OR-Library [1].

This work is conducted in a will to cover many kinds of method such as: local-search, evolutionary and hybrid heuristics. Therefore, this thesis is divided into two parts. The first part gives an extended literature review and the second details the contributions of the work.

The first part is an extended literature review which is composed of chapters 2 and 3. Chapter 2 outlines combinatorial optimisation, knapsack problems, multi-dimensional knapsack problem and so on. This is followed by a literature review on methods proposed to deal with the MKP01 (Chapter 3).

The second part (chapters 4,5,6) presents the contributions of this work. Each of the three chapters introduces a type of solutions to solve the MKP01. Chapter 4 gives the description of two methods based on local-search heuristics, the first, consists in multiple local-search methods adapted for the MU-WDP, while, the second is a new algorithm denoted SLSA produced by the hybridisation of Simulated Annealing (SA) and Stochastic local-search (SLS) to the MKP01. Chapter 5 describes three hybrid approaches that combine population-based and local-search heuristics. The first compares two algorithms produced from the hybridisation of GA with SA and with SLS denoted GA-SA and GA-SLS respectively. The second consists in a memetic algorithm named Memetic Search Algorithm (MSA). MSA is a GA version combined with SLSA presented in Chapter 4. The third approach is

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/

a Self-Adaptive Harmony Search hybridised with Stochastic local-search (SAHS-SLS). Chapter 6 provides another type that consists in hybridising a pre-analysis algorithm with a GA. The purpose here is to extract useful information about the problem and use it to reduce the optimisation charge and time. The proposed Guided Genetic Algorithm (GGA) is a new algorithm that utilises prior-knowledge about MKP01 to calculate high-quality solutions.

Finally, Chapter 7 provides the conclusion on the contributions of this work to the discipline and to the practice, but also its difficulties and limitations and the eventual future work as well.

Chapter 2

Knapsack problems

2.1 Introduction

The 0-1 Multidimensional Knapsack Problem (MKP01) is a NP-hard combinatorial optimisation problem (Martello & Toth, 1990). Furthermore, the MKP01 is a generalisation of the 0-1 Knapsack Problem (KP01) (Wang et al., 2013). In the domain of Combinatorial Optimisation (CO), this problem is widely studied as it has many applications in different domains.

This chapter aims to give a general definition of the main subjects related to the MKP01. The CO, the KP01 as well as the MKP01 are described. The Winner Determination Problem in Multi-Unit combinatorial auctions (MU-WDP) which is a practical application of the MKP01 is detailed as well.

The content of this chapter attempts to draw the border of the addressed problem (i.e. the MKP01). It attempts also to detail the MKP01 in terms of mathematical model, applications and related problems. Thus, the remainder of this chapter is organised as follows: Section 2.2 describes the fundamental topics related to the CO. Section 2.3 gives definitions, mathematical modelling and types of knapsack problem. Section 2.4 details the MKP01 and the MU-WDP. The conclusions and perspectives are given in Section 2.5.

2.2 Combinatorial optimisation

Mathematical optimisation (is often also called optimisation, mathematical programming, numerical optimisation) is the mathematical science of determining the best solutions to mathematically defined problem (Snyman, 2005, p. 2) that satisfy eventually constraints. The optimisation methods are used to solve problems in many domains such as: operational research (Gupta, 2008), graph theory (Kocay & Kreher, 2004), engineering (Rao & Rao, 2009), statistics (Rustagi, 2014), etc.

Combinatorial Optimisation (CO) is a class of problems in which the number of candidate solutions is combinatorial. Each possible solution has an associated cost. The goal is to find the solution with the lowest cost. Because of the vast numbers involved, explicit search an entire search space is not always possible (*Meaning of Combinatorial Optimization*, 2007). A problem is considered as combinatorial optimisation problem if it consists in determining the best solution that satisfies a set of constraints from a finite set of solutions. It is associated with a mathematical *objective function* to optimise and a set of the boundary *constraints* to satisfy.

The importance of the CO is due to a large number of practical applications that can be formulated as a CO problem. Indeed, it has many applications in various fields such as: telecommunication networks design, air traffic scheduling, frequencies allocation of cellular networks, finance, distribution and production planning, resource allocation (Yu, 2013; Cheng et al., 2006).

Although CO problems are often easy to identify, they are usually difficult to resolve. In *complexity theory* most of these problems belong to the class of NP-hard problems (Ausiello et al., 2012). Therefore, up to now, there is no valid effective algorithmic solution able to solve all CO problems.

According to the objective function definition, two categories of CO problems exist *single objective* and *multi-Objectives* problems.

1. Single objective problem is expressed by a single objective function to optimise (minimise or maximise) e.g. maximise a revenue, minimise risks, minimise a time, etc.

2. Multi-objectives problem requires solution according to several objective functions e.g. Maximising income and minimising production cost.

2.3 0-1 knapsack problem (KP01)

The 0-1 Knapsack Problem (KP01) is a NP-hard CO problem. It refers to the common situation of packing the most valuable or useful items without overloading an allowed luggage. The KP01 has been studied for more than a century, as it was first introduced by Mathews (1896). The name "knapsack problem" was first introduced by T. Dantzig (1932). The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorial optimisation, computer science, complexity theory, cryptography, applied mathematics, and daily fantasy sports.

Since its introduction, the KP01 has been the basis of many successful applications such as loading cargo, risk tolerance, budget control, etc. In terms of complexity, it has been shown that the KP01 is a NP-hard problem.

2.3.1 Mathematical modelling

Formally, the KP01 is composed of a knapsack of capacity b and a set N containing n items (variables) j where $N = \{j_1, j_2, \ldots, j_n\}$ each having a value $p_j \in \{p_1, p_2, \ldots, p_n\}$ and a weight $w_j \in \{w_1, w_2, \ldots, w_n\}$. The purpose here is to select the subset of variables (items) x_j that produces the maximum total value with a total weight not exceeding the capacity of the Knapsack. the KP01 can be expressed by the following linear program:

$$KP01 = \begin{cases} maximise & \sum_{j=1}^{n} p_j x_j \\ subject\,to & \sum_{j=1}^{n} w_j x_j \leq b \\ & x_j \in \{0,1\} \qquad j \in \{1 \dots n\} \end{cases}$$

the KP01 is a binary optimisation problem. The decision variables x_j are defined in the set $\{0,1\}$ such that $x_j = 1$ if the item is selected and 0 otherwise. A feasible solution is a solution where the constraints on the maximum weight and the type variables are both verified.

2.3.2 Solution methods

Many solutions were proposed in the previous five decades, especially, during the sixties, seventies and eighties of the last century. The solutions varied between exact, non-exact and hybrid.

The first application of the Branch and Bound method (B&B) for the KP01 is proposed by (Kolesar, 1967). This work was later extended by (Horowitz & Sahni, 1974; Fayard & Plateau, 1975; Nauss, 1976; Martello & Toth, 1977; Pisinger, 1995a). The branching in these approaches is based on an enumeration following a depth-first search of a binary tree (1 means pack the item and 0 otherwise) guided by upper or lower bound (depending on application). This strategy saves memory space and reduces the number of branches. A comparison conducted by (Martello & Toth, 1990) has shown that the B&B method is very effective for small KP01 only. G. B. Dantzig (1957) concluded that the optimal solution of KP (the item may be split) can be obtained by sorting the items in an ascending order obtained by a function expressed by the value and the weight. Items are packed in order until the first item called *critical item* that can not be entirely packed. Its findings have led to the introduction of *core concept* by (Balas & Zemel, 1980), which subsequently, was the source of several other approaches. The *core* is a subset of items around the *critical item* that are considered to determine the optimal solution. The variables (items) outside the *core* are initialised with the value 0 or 1 depending on their ratios value/weight.

Different methods based on Dynamic Programming for solving the KP01 are proposed by (Horowitz & Sahni, 1974; Ahrens & Finke, 1975; Toth, 1980; Rong & Figueira, 2012; Figueira et al., 2013). By (Martello & Toth, 1979) it is shown that, when there is no correlation between the value and the weight, solution by B&B is easier than when the correlation is strong. The dynamic programming is most effective when the correlation is strong, however, the required space increases significantly with the increase in data-size. That means the dynamic programming performs efficiently and within a reasonable time only on small data.

2.3.3 Other related problems

Many specifications of KP have been defined since it was first introduced. Most of them have been inspired from real-world problems. Each particular problem is -itself- a case of research for many projects. Here, we briefly describe some of the KP types.

- *The Continuous Knapsack Problem (CKP)* (Kellerer et al., 2004b). In CKP, the items can be split; in other words; the decision variables x_j are defined in the interval [0, 1]. The CKP can be solved as a linear program with conventional methods such as Simplex (for a polynomial problem). The CKP can be expressed by the following equation:

$$CKP = \begin{cases} maximise & \sum_{j=1}^{n} p_j x_j \\ subject\,to & \sum_{j=1}^{n} w_j x_j \leq b \\ & x_j \in [0,1] \qquad j \in \{1 \ldots n\} \end{cases}$$

- *The Quadratic Knapsack Problem (QKP)* (Gallo et al., 1980). In this type, two related items are taken simultaneously and packed in the knapsack. To express this relationship, a gain value p_{ij} on the connections between each two items (i, j) expresses the objective function (the decision variables are binary) as in the following equations:

$$QKP = \begin{cases} maximise & \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}x_i x_j \\ subject\,to & \sum_{j=1}^{n} w_j x_j \leq b \\ & x_i, x_j \in \{0,1\} \qquad j \in \{1\ldots n\} \end{cases}$$

- *The Bounded and the Unbounded Knapsack Problems* (Kellerer et al., 2004a). Items have copies; if the number of copies is limited this type is called bounded knapsack, if not then we speak about unbounded knapsack. The solution consists of determining the number of copies of each item to pack in the knapsack. In this case, the variables are integer because they consist of the number of items and not whether the item is packed or not (the 0-1 case) and the item cannot be divided (as opposed to the continuous case). Many works have proposed solutions to both problems (Sural et al., 1997; Pisinger, 2000; Poirriez et al., 2009).

- *The two-dimensional Knapsack Problem (BKP)* (Caprara & Monaci, 2004). In this type, the knapsack has two dimensions. It is a version of KP where the model contains two constraints, for example, packing in a knapsack of items having each a weight and a volume. The BKP is often solved as a linear problem with two capacity constraints, as shown in the following mathematical model:

$$BKP = \begin{cases} maximise & \sum_{j=1}^{n} p_j x_j \\ subject\,to & \sum_{j=1}^{n} w_j x_j \leq U \\ & \sum_{j=1}^{n} w_j x_j \leq V \\ & x_j \in \{0,1\} \qquad j \in \{1\ldots n\} \end{cases}$$

- *The MultiObjective Knapsack Problem (MOKP)* (Lust & Teghem, 2012). Several objective functions are optimised. The fact that there are several objective functions needs to find the compromise between them in order to have efficiency. The latter means determining the non-dominated solutions of the problem; i.e. solutions better on all non-dominated objectives by at least one criterion (objective function). A summary on the MOKP is proposed by (Ulungu & Teghem, 1994). The following linear program illustrates a problem of bi-objectives knapsack:

$$MOKP = \begin{cases} maximise & \sum_{j=1}^{n} p_j^1 x_j \\ maximise & \sum_{j=1}^{n} p_j^2 x_j \\ subject\,to & \sum_{j=1}^{n} w_j x_j \leq b \\ & x_j \in \{0,1\} \qquad j \in \{1 \ldots n\} \end{cases}$$

- *The Multiple Knapsack Problem (MuKP)* (Kellerer et al., 2004a). Items are distributed on knapsacks such as an item is assigned to only one knapsack. Therefore, the objective function is to maximise the total value of the items under constraints on the capacity of each knapsack and the items assignment. This issue has been the subject of several works such as (Chekuri & Khanna, 2005; Cotta & Troya, 1998). The model of this type is formulated as follows:

$$MuKP = \begin{cases} maximise & \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \\ subject\,to & \sum_{j=1}^{n} w_j x_{ij} \leq b_i \quad i \in \{1 \ldots m\} \\ & \sum_{j=1}^{n} x_{ij} \leq 1 \qquad i \in \{1 \ldots m\} \\ & x_j \in \{0,1\} \qquad j \in \{1 \ldots n\} \end{cases}$$

- *The Multiple-Choice Knapsack Problem (MCKP)* (Sinha & Zoltners, 1979).

Items are grouped into classes; one item is selected from each class. The constraints are the number of items selected from each group and the total capacity. This problem has largely been addressed (Pisinger, 1995b; Akbar et al., 2006), its model is given as follows:

$$MCKP = \begin{cases} maximise & \sum_{i=1}^{d}\sum_{j=1}^{N_i} p_{ij}x_{ij} & \\ subject\,to & \sum_{i=1}^{d}\sum_{j=1}^{N_i} w_{ij}x_{ij} \leq b & \\ & \sum_{j=1}^{N_i} x_{ij} = 1 & i \in \{1\ldots m\} \\ & x_j \in \{0,1\} & j \in \{1\ldots n\} \end{cases}$$

- *The Quadratic Multiple Knapsack Problem* (Kellerer et al., 2004a). This problem combines between the quadratic and the multiple knapsack problems. In other words, the items are packed in pairs and in multiple knapsacks. The objective function is the maximum amount of profits pairs of items under capacity constraints of each knapsack.

- *The Mixed Knapsack Problem (MiKP)*. In this type, some decision variables are integers. Also, the knapsack can be mixed binary (0-1 mixed knapsack problem), which means that all decision variables are composed of binary and integer variables. This means that some of the indivisible items may have many copies. The following model expresses the Mixed Knapsack Problem:

$$MiKP = \begin{cases} maximise & \sum_{j=1}^{n} p_j x_j & \\ subject\,to & \sum_{j=1}^{n} w_j x_j \leq b & \\ & x_j \in \{0,1\}^n & j \in \{1\ldots n\} \end{cases}$$

- *The 0-1 Multidimensional Knapsack Problem (MKP01)* (Puchinger et al., 2010). This type is composed of a knapsack divided into m dimensions, each with its own capacity, and n items to be packed each with m weights corresponding to the knapsack dimensions and a value. the MKP01 is detailed in the next section.

2.4 The 0-1 multidimensional knapsack problem (MKP01)

Various names have been used to describe the 0-1 Multidimensional Knapsack problem (MKP01) including for example Multi-knapsack problem, m-dimensional knapsack problem, multi-constraints knapsack Problem 0-1, etc. Historically, the problem is relatively old as its mathematical model dates from the mid-fifties. However, it remains among the most known problems in the field of CO. Its importance lies in the fact that a significant number of problems of daily-life were modelled in its form. Indeed, the MKP01 was applied to solve the projects selection problem (Beranek, 1965; Petersen, 1967; Mcmillan & Plane, 1973; Meier et al., 2001).

the MKP01, is used to model various applications such as decision-making process, set packing (Fox & Scudder, 1985), combinatorial auctions (Rothkopf et al., 1998; De Vries & Vohra, 2003), resource allocation (Johnson et al., 1985), project selection (Petersen, 1967), cutting stock (Gilmore & Gomory, 1966), cargo loading (Shih, 1979), asset-backed securing (Mansini & Speranza, 2002), manufacturing in-sourcing (Cherbaka et al., 2004), computer systems design (Ferreira et al., 1993), and capital budgeting (early examples include (Markowitz & Manne, 1957; Lorie & Savage, 1955; Weingartner, 1966)).

2.4.1 Mathematical modelling

The MKP01 is composed of n items and a knapsack with m different capacities c_i where $i \in \{1,\ldots,m\}$. Each item j where $j \in \{1,\ldots,n\}$ has a profit p_j and can take w_{ij} of the capacity i of the knapsack. The goal is to pack the items in the knapsack so as to maximise the profits of items without exceeding the capacities of the knapsack. the MKP01 can be represented as the following integer program:

$$MKP01 = \begin{cases} maximise & \sum_{j=1}^{n} p_j x_j & \\ subject\,to & \sum_{j=1}^{n} w_{ij} x_j \leq c_i & i \in \{1 \dots m\} \\ & x_j \in \{0,1\} & j \in \{1 \dots n\} \end{cases}$$

A feasible solution X for the MKP01 represents the selected items to be packed in the knapsack. The decision variables x_j are binary where $x_j = 1$ means that the item j is packed in the knapsack, and $x_j = 0$ means that item j is not packed in the knapsack. w_{ij} represents the weight of the item j on the dimension i.

2.4.2 The winner determination problem in multi-unit combinatorial auctions (MU-WDP)

The Combinatorial Auction (CA) is an efficient mechanism in which a seller is faced with a set of price offers for various bundles of goods. The aim is to allocate the goods in a way that maximises the auctioneer revenue. In multi-unit combinatorial auctions, each good has a set of occurrences. The buyer instead to bid for a set of goods, bids for a set of bundles of goods with a price. The multi-unit winner determination problem (MU-WDP) consists of selecting the subset of bids which maximises the revenue of sellers.

The combinatorial auction is a mechanism that has proven its effectiveness in solving many problems such as resources allocation in multi-agent systems, the auctions for radio spectrum rights (Caplice, 1996), airport slot allocations (Jackson, 1976), transportation services (Jones, 2000; Graves et al., 1993), course registrations (Rassenti et al., 1982), and commercial time slot allocations (Sheffi, 2004).

In multi-unit combinatorial auctions, for each good i, we have some number of units q_i of this good for sale, contrary to the single-unit combinatorial auctions where there is only one unit of each good.

For example, suppose there are three goods g_1, g_2 and g_3. We have 10 units of good g_1, 4 units of good g_2 and 5 units of good g_3 for sale. We have the following bids:

- $\{(8g_1, 4g_2, 2g_3); 20\}$
- $\{(2g_1, 2g_2, 4g_3); 10\}$
- $\{(5g_1, 1g_3); 8\}$

This means, for example, that the first bidder wants to buy 8 units of g_1, 4 units of good g_2 and 2 units of g_3, and he is willing to pay 20 for this.

In general the problem can be stated as follows:

Let us consider G a set of m goods, $G = \{g_1, g_2, \ldots, g_m\}$ to be auctioned, and l_i denote the number of available units of good i for sale. Let us consider B a set of n bids, $B = \{B_1, B_2, \ldots, B_n\}$. A bid $B_j = [(S_{1j}; S_{2j}, \ldots S_{mj}); P_j]$ where $S_{kj} \geq 0$ is the requested number of units of good k, and P_j is the price of B_j, ($P_j > 0$). Finally the decision variables are defined as follows: $x_j = 1$ if the bid B_j is accepted (a winning bid), and $x_j = 0$ otherwise (a losing bid). The WDP in multi-unit auctions can be modelled as the following integer program:

$$MU-WDP = \begin{cases} maximise & \sum_{j=1}^{n} p_j x_j \\ subject\ to & \sum_{j=1}^{n} S_j^i x_j \leq mu_i \quad i \in \{1 \ldots m\} \\ & x_j \in \{0,1\} \quad\quad\quad\ \ j \in \{1 \ldots n\} \end{cases}$$

the MU-WDP is the problem of finding an allocation of winning bids that maximise the auctioneer's revenue under the constraint that for any good i, the sum of units of i over all the winning bids does not exceed mu_i.

2.4.3 Solutions for the MU-WDP: breve state-of-the-art

Several approaches have been proposed for the single-unit combinatorial auctions version where the number of goods for each bid is exactly equal to one, among these approaches, we cite: Branch on Bids (BoB) (Sandholm & Suri, 2003), Combinatorial Auctions BoB (CABoB) (Sandholm et al., 2005), Branch-on-Price (Kameshwaran & Benyoucef, 2006), Branch-and-Cut (Escudero et al., 2009), Combinatorial Auction Structured Search (CASS) (Fujishima et al., 1999). Most of these solutions are based on the Branch-and-Bound searching method. Other methods are investigated for the single-unit winner determination problem such as the dynamic programming (Rothkopf et al., 1998), the linear programming (Nisan, 2000), the integer programming (Andersson et al., 2000), the constraint programming to solve the Vickrey combinatorial auction (Holland & O'sullivan, 2004).

In order to handle large scale instances of the winner determination problem, the non-exact methods are often preferable. The heuristic methods are then largely used. We can cite the local search approaches for example Hybrid Simulated Annealing (SAGII) (Guo et al., 2004; Guo, Lim, Rodrigues, & Zhu, 2006), Casanova (H. H. Hoos & Boutilier, 2000), the stochastic local search (Boughaci et al., 2010) and the mimetic algorithm (Boughaci et al., 2009). We can also find other approaches like Artificial Fish-Swarm Algorithm (AFSA), the ant colony algorithm (Zheng & Lin, 2012) and the genetic algorithm hybridised with the ant colony algorithm (L. Chen et al., 2008).

Contrary to single-unit WDP, there are few works proposed to solve the multi-unit the MU-WDP. Among them, we can find a Branch-and-Bound based algorithm (Gonen & Lehmann, 2002), Combinatorial Auctions Multi-Unit Search CAMUS (Leyton-Brown et al., 2000) which is a version of CASS adapted to the multi-unit case. An algorithm based on the Particle Swarm Optimisation (PSO) approach (Farzi, 2010), a local search algorithm (Singh et al., 2012), a heuristic suggestion based on a Lagrangian relaxation (Guo, Lim, Rodrigues, & Tang, 2006).

2.5 Conclusion

The aim of this chapter was to draw the borders of the 0-1 multidimensional knapsack problem (MKP01) and locate it in the cloud of combinatorial optimisation problems. It was shown that the MKP01 is one of the types of knapsack problem KP. Specifically, it is a generalisation of the 0-1 knapsack problem (KP01). The MKP01 is a well-known NP-complete CO problem widely studied in the field of combinatorial optimisation. Moreover, various applications can be formulated as MKP01. The winner determination problem in multi-unit combinatorial auctions is one of these applications. The next chapter will cover the methods in literature proposed to solve the MKP01.

CHAPTER 3

0-1 MULTIDIMENSIONAL KNAPSACK PROBLEM: OVERVIEW

3.1 Introduction

Literature is rich with approaches proposed to solve the 0-1 Multidimensional Knapsack Problem (MKP01). Indeed, several studies consider MKP01 as a Combinatorial Optimisation (CO) problem sufficiently complex to test and validate their approaches. This justifies the importance given to its solution. To be able to contribute to this domain, it is important to have a comprehensive idea on how researchers tried to deal with this problem. In this context, this chapter gathers and classifies works dealing with the solution of MKP01.

Thus far, a number of studies have summarised the state-of-the-art of MKP01, some including all approaches (Hanafi & Freville, 1998; Chu & Beasley, 1998; Gottlieb, 1999; Osorio et al., 2002; Fréville, 2004; Raidl & Gottlieb, 2005; Kellerer et al., n.d.; Fréville & Hanafi, 2005), whereas others included only works applying one particular method, such as Tabu Search (TS) (Hanafi & Freville, 1998) or Evolutionary Algorithm (EA) (Gottlieb, 1999). Fréville (2004) classifies the methods into exact and heuristic methods and describes the various theoretical analyses of MKP01 and the existing experimental data. Varnamkhasti (2012) classifies methods into exact, heuristic and meta-heuristics.

MKP01 solution methods can generally be classified into two categories:

1. The deterministic methods (exact) include all methods that are used to calculate the optimal solution.
2. The approximate methods (non-exact) gather the methods that do not guarantee to obtain the optimum solution, but return an approximate solution near the optimality.

3.2 Deterministic methods

Mathematical theories aim to simplify physical and natural phenomena in order to study, understand and resolve them. Researchers such as Church, Turing, Babbage, Von Neumann, Dantzig etc, installed the theories and the fundamentals of computing and algorithms on mathematical foundations. For the problem of MKP01 mathematical methods were obviously used to solve it. In the following, we describe the main methods.

3.2.1 Explicit enumeration

Before the emergence of approximate algorithms, the trend was focused on developing algorithms with the main idea to enumerate all the possible solutions in order to find the best. The proposed approaches tried to introduce mechanisms allowing to avoid revisiting the solutions already visited, in order to save effort and time. Balas (1965) as Geoffrion (1969) and Glover (1965) proposed enumerating algorithms for solving linear binary programs in general and MKP01 as an example (see also (Balas, 1967)). Then, Lemke and Spielberg (1967), Trauth Jr and Woolsey (1969) and Breu and Burdet (1974) conducted experiments to examine the effectiveness of these methods. Their and others' results revealed that enumeration methods may be outperformed by others more effective methods such as Branch and bound (B&B). Therefore, these methods are more unlikely to deal effectively with big-size MKP01. Furthermore, their processing time could not be polynomial. The introduction of more sophisticated algorithms becomes necessary to solve large problems. To respond to this need, methods based on the lower and upper bounds were invented, such as the branching methods (e.x. B&B). These methods are able to limit the search process to a part of the solution space, and therefore make them gaining valuable time.

3.2.2 Implicit enumeration

The Branch and Bound (B&B) method applied to the MKP01 is a process based on the enumeration, using a binary tree of feasible solutions built based on a Lower Bound (LB) or Upper Bound (UB). The first researcher that adapted B&B to the MKP01 is Thesen (1975), using the linear relaxation of the MKP01. But this relaxation is often applied to the KP, as it consumes a significant memory space, however, the produced results were not encouraging. Practically, the first succeeded B&B to MKP01 came four years later by (Shih, 1979). A LB is calculated considering the linear relaxation of each constraint apart, then, the solution is calculated on the basis of these partial solutions. The results of its application on a set of test data, has shown its effectiveness. It also demonstrated its superiority over the algorithm by (Balas, 1965) in terms of processing time (it was ten times less than the Balas' approach). (Gavish & Pirkul, 1985) claimed that the limitation of Shih's method is in the large memory space required, in addition to its weakness to give good results when applied to larger problems.

3.2.3 Dynamic programming

Dynamic Programming (DP) was presented for the first time by (Bellman, 1954). It is a simple enumeration algorithm invented primarily for solving combinatorial problems. It is based on cutting the original problem into a set of partial sub-problems easier to solve. After solving all the sub-problems, the partial solutions are combined so as to build the overall solution of the problem. Toth (1980) is the first who adapted the DP to KP. This was followed by (Pisinger, 1997) with a DP algorithm to build a collection of variables from the original problem named *Core*. Plateau and Elkihel (1985) combined DP with an enumeration algorithm. Similarly to the Primal and Dual Greedy algorithms (see Section 3.3.1), Weingartner and Ness (1967) presented two approaches based on DP. The first, fill a knapsack initially empty, while the second begins with a knapsack containing all the items. A comprehensive study on the DP application to the KP problem has been conducted by (Pisinger, 2005).The works by (Marsten & Morin, 1977, 1978) are the first applications of the DP to solve MKP01. The approach they proposed extends the algorithm by (Gilmore & Gomory, 1966) on KP by integrating DP and B&B. In addition, algorithms combining a pretreatment that calculates the bounds and DP were presented by (Balev et al., 2008). Through the experimental results, it may be concluded that this approach reduces the solution time and requires low memory space. Boyer et al. (2009) have chosen to deal with DP surrogate relaxation. Two versions were presented in this work. In the first, DP is applied to the surrogate relaxation of MKP01; whereas, in the second, the first version is combined with the Branch and Cut method (B&C) to improve the bounds. The results showed an increase in terms of optimisation time and results.

3.2.4 Relaxation

Relaxation is a transformation of the mathematical model of a CO problem. It consists to remove or reduce the constraints to simplify solving the model. A penalty charges made on the objective function are added to compensate the deleted constraints. The solution of the relaxed problem represents a lower bound to the original problem. Various relaxations have been proposed to MKP01 such as Linear relaxation, Lagrangian relaxation, Surrogate relaxation etc. With the introduction of this method, many researchers were interested in its application for solving combinatorial problems. This coincided with a diversification and a large emergence of heuristics.

3.2.4.1 Linear relaxation

The Linear Relaxation of MKP01 is perhaps the simplest among the relaxation methods. It involves replacing the constraint $x \in \{0,1\}$, by $x \in [0,1]$, in another word, consider the decision variables as fractional values instead of binary which means an item may be split and packed in the knapsack. Simplex method is often used as a solution tool for the linear relaxation of MKP01, and the solution is used as LB for solving MKP01.

As mentioned before, Dantzig invented Simplex for solving linear programs and among them, of course, the relaxed KP01. The solution of a linear relaxed problem is an UB for the original problem. From solving KP01 by relaxation, this method has been generalised to MKP01. Kochenberger et al. (1974) presented a heuristic based on the MKP01 linear relaxation to calculate a feasible solution that is improved later. Fréville and Plateau (1996) suggested several algorithms specifically for the bi-dimensional knapsack problem most of them based on linear and surrogate relaxation.

The most successful method based on the MKP01 linear relaxation is Pivot and Complement (Balas & Martin, 1980). It is a heuristic that combines a version of Simplex and a complementary algorithm. It begins by relaxing the MKP01, then, it performs a bounded variables Simplex with a series of Pivot, which is used to enter the bounded variables in the base with the lowest cost. In a second step, the obtained solution is improved and adjusted to a feasible solution to the original problem. The Pivot movements are applied according to a priority list. This method has been proved to be efficient after its test on several problems different in complexity and data size. Pivot and Complement is a component for many approaches, one of the most successful is the approach presented by (Aboudi & Jörnsten, 1994) where this method is combined with TS. The algorithm by (Løkketangen et al., 1994) provides a solution to improve its efficiency by replacing the list of priority by a TS algorithm. This, has been successfully tested on relatively simple test data set.

3.2.4.2 Lagrangian relaxation

The Lagrangian relaxation of MKP01 merges the structural constraints on the maximum weight and the objective function and defines the Lagrangian multipliers as new variables. Usually, the number of new variables is significantly less than those of the original problem. Due to the constraints removal, a penalty expression is added to the objective function. Therefore, solution of the problem is simplified from determining n variables to find only m multipliers that optimise the new objective function.

The effectiveness of Lagrangian relaxation compared to Linear relaxation has attracted more attention. Obviously, it is easier to find a small number of multipliers to solve a complex problem with complex methods. Although the solutions are not optimal, it is preferred to deal with data of large size. According to how the approaches use the Lagrangian relaxation method, two categories are distinguishable. First, approaches that use it to calculate an UB, usually use the dual MKP01. This is integrated into a solution with a branching method (usually B&B) in order to find the optimal solution. Second, other approaches use it to directly calculate an approximate feasible solution to the MKP01. It is important to note that in the first category, the quality of results depends on how close is the UB to the optimum, which depends on the heuristic used for its measurement. An UB far from the optimal means that a long time would be needed if the data are big. The closer is the UB the shorter is the processing time. In the second category, the results obtained by the existing approaches, are very eligible but not competitive with other methods less complicated and faster.

Magazine and Oguz (1984) extended the Dual Greedy (Toyoda, 1975) by applying it to find the Lagrangian multiplier. In their algorithm, the variables are initialised to 1, then one multiplier is modified at each iteration to have a feasible solution. Later, Volgenant and Zoon (1990) improved this approach and, instead of changing one multiplier at once, they suggested changing k multipliers ($k > 1$). After several experiments using data generated randomly, they concluded that this change has further refined the first approach, but with a longer optimisation time. Yoon et al. (2005) proposed LM-ES (Lagrangian Multipliers-Evolutionary Search), an evolutionary algorithm that calculates the multipliers of the Lagrangian relaxation of MKP01. This, by developing a different Lagrangian relaxation and its integration with ES to calculate the UB and LB of the dual MKP. The results showed that the proposed relaxation is better, however, it took longer time to achieve good solutions. Yoon et al. (2012) proposed the FPLS relaxation. It is a heuristic based on the Lagrangian Capacity. The proposed relaxation allows obtaining better multipliers than the Lagrangian relaxation. The FPLS relaxation is recently integrated with the Ant Colony Optimisation method (Nakbi et al., 2015). All these approaches are trying to calculate the best multipliers that give the best solutions to MKP01.

3.2.4.3 Surrogate relaxation

Other relaxation techniques have also been proposed to solve various CO problems and have been applied to MKP01 such as the surrogate and the decomposition relaxations.

Surrogate relaxation was introduced by (Glover, 1965). It means to replace the constraints by only one constraint called Surrogate constraint. From the state of the art, it is noticed that the Surrogate relaxation has been used with deterministic and approximate heuristics to solve the MKP01. It has also been combined with other approaches as a pretreatment procedure to reduce the problem size.

The surrogate relaxation brings a great simplification of combinatorial problems, which caused its great emergence during the past three decades. It is better than the Linear and the Lagrangian relaxations in terms of simplicity and efficiency which explains the great interest that has given to it. The Surrogate relaxation is combined with various heuristics to solve MKP01. This provided effective approaches that have managed to solve difficult instances of the problem. Greenberg and Pierskalla (1970) completed Glover's work and conducted the first major study of Surrogate relaxation in the programming area. Glover (1968), Karwan and Rardin (1979) and Dyer (1980) enriched this work by more problems and data. Fréville through his works claimed that when the Surrogate relaxation is used to calculate a LB to MKP01 problem, it can obtain better results than the methods in the literature. This conclusion was supported by (Gavish & Pirkul, 1985) through a B&B method based on a LB calculated by Surrogate relaxation. This direction is a promising field of study, that means there is still a margin of improvement for the LB to achieve. Furthermore, in order to improve the LB, some researchers opted for the dual surrogate relaxation, however, the solution of Dual Surrogate itself is a NP-complete problem (Boyer, 2007). Several approaches adopting this approach including the work by (Dyer, 1980; Karwan & Rardin, 1979; Sarin et al., 1987). Freville and Plateau (1986) developed three algorithms based on the surrogate relaxation for solving the MKP01. They integrated techniques to fast repair, disruption and define the strong variables. It has been revealed that this approach requires a long time, so it is more suited to the problems of small or medium size.

3.3 Heuristics and meta-heuristics

In this part, two main classes of approximate solutions are considered, the heuristic and the meta-heuristic methods. Here, heuristics are different than meta-heuristic as they do not contain a technique to avoid the local optimum.

3.3.1 Heuristics

The Gradient algorithm (Cauchy, 1847) was the source of inspiration for inventing the Greedy algorithm. The Greedy algorithm was firstly proposed by (G. B. Dantzig, 1957) for dealing with the KP. A few years later, it has been adapted to the KP01 (Senju & Toyoda, 1968). The *Effective Gradient Greedy* (Senju & Toyoda, 1968) is a heuristic that may be summarised in three steps. Firstly, calculate an *efficiency* value for each item (in the general case an item is a variable of the KP01 mathematical model). It is a mathematical relationship (Eq. 3.1) based on the item's value and weight. Secondly, sort the items decreasingly according to their efficiency values, in a way that the best items will be prioritised. Finally, pack one by one in order, the items in the knapsack, while all the constraints are verified. By abstraction, adding an item to the knapsack is expressed by setting its representative variable to 1. As this algorithm built a solution from an empty knapsack, it was named *Primal Greedy* in a sense that all variables are initially equal to 0.

$$e_j = \frac{p_j}{\sum_{i=1}^{m} w_{ij} \sum_{j=1}^{n} w_{ij} - b_i} \tag{3.1}$$

The Primal Greedy algorithm was tested on MKP01 data-sets, relatively complex if we consider the computational power of this period (60 items and 30 dimensions). As items are selected in order according to their positions, the efficiency function determines the quality of the solutions obtained by the algorithm.

Therefore, other formulations of this function have been suggested, such as the simple expression Dobson (1982), another more efficient and complex was introduced by (Freville & Plateau, 1994) or using the dual MKP01 problem presented by (Puchinger et al., 2006). Kochenberger et al. (1974) generalised the Primal Greedy to the MKP in which the variables are defined in the interval [0, 1].

Toyoda (1975) introduced the *Dual Greedy* algorithm. This new version starts the optimisation with an infeasible solution, where variables are all initially equal to 1 (i.e. all items are initially in the knapsack). Then, sorts the items in an ascending order according to their efficiency values. After that, it iteratively removes one item at once by setting their variables to 0 and repeats this while at least one constraint is still not satisfied . It was proven by experimental comparison that Dual Greedy is more effective than Primal Greedy. Furthermore, Dual Greedy is successfully integrated to repair infeasible solutions in the Genetic Algorithm by (Chu & Beasley, 1998).

3.3.2 Meta-heuristics

Despite their effectiveness in solving difficult combinatorial problems, heuristic quickly converges to local optima. Meta-heuristics are a logical and a necessary evolution of heuristics to deal with this limitation. Indeed, meta-heuristics include diversification mechanisms allowing them to change the search area to cover the entire space of feasible solutions and avoid the local optimum. For dealing with large data, Meta-heuristics are better than heuristic and deterministic approaches because they are able to reach solutions very close to the optimal within a reasonable time.

Many meta-heuristics have been proposed to solve MKP01, such as Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA), Ant Colony Optimisation (ACO), Particle Swarm Optimisation (PSO), Harmony Search (HS), etc.

The meta-heuristics can be classified according to the structure and the operating principle into two groups: *Single solution-based* and *Population-based*. Single solution-based methods build a single solution that will be improved by browsing its neighbourhood in the solution space through random or semi-random movement. Population-based methods initialise a population of solutions and improve it by iterative operations and extract the best solution at the end of the optimisation process. In the sections that follow, we cite the most popular meta-heuristics that have been the most successful in solving the MKP01 and this in chronological order.

3.3.2.1 Single solution-based meta-heuristics

Several methods may be included under this class of solution. The Simulated Annealing (SA) and the Tabu Search (TS) are considered in this work.

1. **Simulated Annealing**

   The Simulated Annealing (SA) invented by (Kirkpatrick et al., 1983) is a local search algorithm that simulates the method used by miners, for cooling the metal gradually and periodically by decreasing the temperature until it (metal) takes its stable state. Few studies have used SA to solve MKP01 despite being old and more effective in such combinatorial problem than other heuristics such as Hill-Climbing. It Mostly hybridised with a population-based method.

   SA can be decomposed into two phases. The first phase consists in preparing the data by generating an initial feasible solution X. The second phase is the optimisation of the initial solution. This operation is based on the temperature parameter T fixed initially as T_0.

   The optimisation includes four steps given as follows:

a) Creating a neighbour solution X' of X. For that, one item I is chosen arbitrary. If I increases the objective function $f(X)$ then it will be accepted i.e. added to the solution X', otherwise it will be added if the comparison $\exp^{-(\Delta f()/T)} > R$ is true. Where R is a random value and $\Delta f() = f(X') - f(X)$.

b) The first step may cause a conflict. In order to eliminate all conflicts, and make X' as a feasible solution, it is repaired by removing an item. The item to be removed is chosen in two manners according to the probability $P(P = 0.7)$. Either an item is randomly chosen or the worst one in X' is found and removed. This process is repeated until the elimination of all conflicts.

c) During the process, among the neighbours, every solution that increases the objective function is considered as the best solution and saved in X^*. By the end of the process X^* contains the best solution.

d) The last step in this phase consists in updating the temperature value using the Coefficient of Temperature CT.

The optimisation phase is repeated for a certain Number of Iterations (NI) fixed empirically.

Drexl (1988) is the first to propose the application of SA to MKP01. Practically, this version of SA performs two random exchanges at each iteration to go from one solution to another similar with keeping its feasibility. SA has been improved to a new version named Threshold Accepting (Dueck & Wirsching, 1989). With this version, they obtained better results than Drexl (1988). Qian and Ding (2007) augmented SA with a Markov chain that determines when the temperature should be decreased. This allows estimating the execution time of the algorithm, i.e. ensure that after an optimisation time the process stops. Its application showed that it was faster than SA with its strategy of cooling but with poor performance. Battiti and

Tecchiolli (1992) compared SA with Tabu Search (TA) and the comparison showed a total superiority of TS over SA. Perhaps the biggest obstacle of using SA is how to determine the right value of the initial temperature and what cooling strategy to adopt. These two factors, although determinants, their ideal values depend on the addressed problem and may differ from one to another.

2. **Tabu Search**

   Tabu Search Algorithm (TS) was invented by (Glover, 1986, 1989, 1995). This approach is one of the most popular meta-heuristics that has acquired success in a wide variety of CO problem (Glover & Laguna, 2013). The main innovation that has introduced in TS is the *tabu list*. It is an adaptive memory allowing a sensitive exploration as a basis for solving combinatorial problems. This memory is divided into short and long-term memories.

   Chronologically, the first application of TS for the MKP01 was conducted by (Dammeyer & Voß, 1993). It was developed on the basis of the Reverse Elimination Method (REM) (Glover, 1989). This work proposed and compared the static and the dynamic management strategies of tabu list. The obtained results using the data-set by (Drexl, 1988) showed the dynamic strategy as the best version which also succeeded to surpass SA. Furthermore, the REM has reached the optimal solution in 44 of the 57 experimental problems.

   A strategy of oscillation between the solutions' feasibility borders was presented by (Glover & Kochenberger, 1996). In this version of TS, the search process is allowed to go into the regions of infeasible solutions to diversify the search. Good results have been obtained by applying this approach on many test data including the 57 problems by (Drexl, 1988); these have all been resolved to optimality.

   A TS version combined with Greedy algorithm and guided by Surrogate constraints was introduced by (Hanafi & Freville, 1998). The Surrogate constraints introduced by (Glover & Kochenberger, 1996) are calculated to reduce the problem size. The TS process is divided into two phases, constructive and destructive and adopts Greedy algorithms to perform the movements in the feasible and infeasible solutions. The movements to infeasible solutions follow an oscillation strategic dynamic and adaptive to the addressed problem. This approach has been applied to a variety of data and achieved very competitive results.

The TS hybridisation is also one of the approaches adopted by researchers. In this sense, Vasquez and Hao (2001) combined TS with Simplex. This former is partially applied to solve the linear relaxation of MKP01, then, TS improves the obtained solution. The results were satisfying for certain instances, however, the processing time was long.

Motivated by assuming that the search space around x (the optimal solution of the linear relaxation MKP01) should contain high-quality solutions, Vasquez et al. (2001) proposed a two-phase approach. The first phase solves a linear relaxation of MKP01 by the Simplex method. The second phase carefully explores some areas around this fractional solution using TS. An additional constraint $\sum_{j=1}^{n} x_j = k$ allows research on a small limited number of a partial solution was included. This approach has been examined on some data-sets (Drexl, 1988; Chu & Beasley, 1998; Glover & Kochenberger, 1996). This TS version was able to improve the most part of the results. Other researchers has focused on the calculation of effective values of k such as (Glover, 1965; Fréville & Plateau, 1993; Vasquez & Vimont, 2005). Several results have been improved by extending TS version presented by (Vasquez et al., 2001) especially that proposed by (Vasquez & Vimont, 2005).

3.3.2.2 Population-based meta-heuristics

1. **Genetic Algorithm**

   Thiel and Voss (1994) estimated that the solution of MKP01 with the standard GA is not able to provide good solutions when it comes to complex problems. They offered a more promising approach by combining GA with TS. This hybridisation approach was able to significantly improve the performance of GA. However, the obtained results were not competitive with other heuristics.

   Cotta and Troya (1998) proposed a hybrid version of GA in superposition with the Greedy algorithm. Although this approach is guided by knowledge of the problem, moderate results were obtained by applying these approaches on small data.

   In the GA version proposed by (Khuri et al., 1994), the approach enriched the population of chromosomes (solutions) by infeasible solutions and defined a fitness function which introduces the concept of penalty s (s is described by the total amount of superfluous knapsack dimensions). The approach has been tested on a few simple data and moderate results were reported. This confirms the findings on the poor performance of standard GA on MKP01 claimed by (Thiel & Voss, 1994).

   In the GA version proposed by (Rudolph & Sprave, 1996) for solving MKP01, two main contributions were included. First, the selection operator chooses only neighbouring solutions. Second, a calendar manages the frequency and the offspring acceptance rate (new individuals created). As in (Khuri et al., 1994), The infeasible solutions were accepted.

   Hoff et al. (1996) tried to prove that GA could be effective with a set of appropriate parameters. In this sense, they tested many different operators and settings to determine the best. Based on their conclusion, they were able to express using the number of variables (i.e. items) n, the desirable parameters for a good GA as follows: an initial population of size $5 * n$ containing only feasible solutions, a number of generation of $250 * n$, a selection operator according to the fitness with the Monte Carlo method, a crossover in Burst with a probability of 0.5 and a reverse mutation with a probability of $1 \div n$. Despite the promising results, there is no formal method to proof that these settings produce always good solutions, mainly because the proposed version has been tested only on a simple problems of small size.

Chu and Beasley (1998) suggested a version of GA for MKP01 solution. So far, this work is among the algorithms that achieved the most promising results. This version of GA combines a standard GA and a strategy for repairing the infeasible solutions. This former sorts the items in the descending order according to their efficiency value (a ratio between the value and the weight). The repair strategy repeatedly excludes an item (in order) until the solution becomes feasible. In addition to the already existing test data, The experimental scope introduced others more complex data-sets. These data are the best-known and the widely used by researchers. A comparison with the CPLEX Optimiser was also conducted. Encouraging results have been obtained, however, the execution time is the major inconvenient of this approach.

Djannaty and Doostdar (2008) suggested a hybrid GA (Hybrid Genetic Algorithm HGA) in which the initial population is created on the basis of the work by (G. B. Dantzig, 1957) on the KP01. It also adopted the penalty function approach to avoid the strong constraints and expresses the fitness function. However, HGA has been tested only on simple data and uses a binary codification of the chromosomes which requires unnecessary memory space.

The Orthogonal Genetic Algorithm (OGA) is a version of GA that incorporates the orthogonal geometric concept. This technique allows to virtually and dynamically locate solution over its neighbouring solutions, in order to consider them or even adopt them. This method has been implemented via the structures that manage solutions neighbour relations. They were integrated into GA either partially to control an operator or globally as a full operator. In this sense, Li et al. (2006) proposed an OGA to tackle the MKP01 incorporating a crossover operator which forms a sample of the best genes among the offspring from which it generates and repairs offspring of higher quality.

Gallardo et al. (2005) suggested a cooperative approach between an Evolutionary Algorithm (EA) and B&B. Both methods Exchange useful information as follows: EA provides the lower bound (LB) to B&B to significantly reduce its research area. In parallel, the solution of a partial optimisation by B&B serves to guide EA to areas of promising solutions. The hybrid algorithm that results has been tested on large instances of the MKP01 problem with encouraging results.

The approach defined by (Alonso et al., 2005) applies a GA to estimate a set of multipliers of a surrogate relaxed MKP01. Then, it performs an EA combined with a Local Search applied periodically to improve all individuals of the population. Good results were obtained with this method.

2. **Harmony Search**

   Harmony Search (HS) (Geem et al., 2001) is a new population-based heuristic that stimulates the construction of a musical harmony from a repertory (named Harmony Memory (HM)) containing other harmonies. The HM is composed of Harmony Memory Size (HMS) harmonies coded by vectors representing feasible solutions. Each iteration, a new harmony is improvised by adding an item either randomly or from the HM according to the harmony memory considering rate (*HMCR*). Then, the item may be adjusted by a

Bandwidth (bw) according to the pitch adjusting rate (PAR). Finally, the improvised harmony replaces the worst harmony in HM, if its fitness is better. Since its appearance, HS has known several changes focusing mainly on various schemes of components HMCR , PAR and BW.

In the context of our work, several approaches have proposed the solution of MKP01 by HS. In the NBHS (New Binary Harmony Search) (X. Kong et al., 2015), the authors have chosen the binary representation of the harmonies, replaced the adjustment PAR by a step similar to the GA mutation operator, introduced an Adaptive HMCR and proposed the generation of the new harmony from a selection of elements named Mean Harmony (MH). The conducted tests showed that this approach converges slowly but effectively towards solutions close to optimal solutions. Despite this, the structure of NBHS differs significantly from the original HS. In HHS (Hybrid Harmony Search), B. Zhang et al. (2015) combined the exploration of HS and the exploitation of the FFO local search (Fruit Fly Optimisation). HHS has a new scheme of HMCR giving it more exploration ability. At each iteration, an intensification by application of FFO is performed at each harmony to explore its neighbourhood, and improve its quality. Experimental tests on complex instances show that this approach performs better than other HS-based methods, however, from a number of generation (100,000) and the complexity of HHS it can be concluded that the execution time is relatively long.

Many researchers have attempted to integrate the theories of a quantum computer in heuristic approaches. Quantum computing is a new generation of computers (CPUs) that exploits characteristics and properties such as entanglement and superposition to represent information. The bit, which takes state 0 or 1 is replaced by the Qubit which is used to encode more than two states. This technology is qualified as the future of computer, there are a few implementations such as Google [1] 100 million times faster than normal PC, NASA[2] developed by the Canadian company D-Wave [3] or IBM set free on its Cloud [4] platform. Layeb (2013) integrated some of its theory such as Interference and Superposition operators in his QIHSA (Quantum Inspired Harmony Search Algorithm). It offers a quantum representation of harmonies, a quantum mutation and another operator. Tests on some MKP01 data by (Chu & Beasley, 1998) m = 5-10 and n = 100 showed a moderate performance of QIHSA.

A judicious choice of the HMCR, PAR and BW values may induce a good performance of HS. For this reason, the major part of the research on the HS have focused on the invention of mechanisms to set these three parameters. DGHS for KP01 presented by (Xiang et al., 2014) integrates a dynamic generation of PAR and HMCR such as:

$$\begin{cases} HMCR(t) & = & HMCR_{min} & + & \frac{HMCR_{max}-HMCR_{min}}{NI} \\ PAR(t) & = & PAR_{min} & + & \frac{PAR_{max}-PAR_{min}}{NI} \end{cases}$$

Where NI is the number of iteration and t the number of current iteration). It iteratively improvises a harmony either by selecting an item from the best harmony or by selecting a harmony randomly in HM. Knowing that in the second option, it is undergoing an adjustment by binary mutation with a PAR probability. To restore any infeasibility, DGHS has a two-phase method DROP and ADD.

3. **Other methods**

[1] https://www.technologyreview.com/s/544421/googles-quantum-dream-machine/
[2] https://ti.arc.nasa.gov/tech/dash/physics/quail/
[3] http://www.dwavesys.com/
[4] http://www.research.ibm.com/quantum/

- *Ant colony Optimisation (ACO)*. Recently, M. Kong et al. (2008) have presented an Ant Colony Optimisation (ACO), called Binary Ant System (BAS) for the MKP01. BAS uses a pheromone deposit method specific to the case of binary solutions and allows the generation of feasible solutions during the optimisation procedure. BAS has been compared to three other proposed ACO algorithms by (Leguizamon & Michalewicz, 1999; Fidanova, 2002; Alaya et al., 2004), on 60 instances of the data sets by (Chu & Beasley, 1998). BAS obtained all the best solutions that are not met by the other three algorithms. But the remain 210 instances have not been used. ACO was also used by (Ke et al., 2010) for solving MKP01.
- *Variable Neighbouhood Search (VNS)*. Puchinger and Raidl (2008) presented a version of the variable neighbourhood search algorithm. Unlike the standard VNS, the proposed solution (Relaxation Guided Variable Neighbourhood Search) is based on different methods to adaptively order (compared to the solution) the Neighbourhood of a solution.
- *Particle Swarm Optimisation (PSO)*. M. Kong and Tian (2006), and Hembecker et al. (2007) presented the optimisation by particle swarm optimisation (PSO) method for the MKP.
- *Scatter Search (SS)*. Hanafi and Wilbaut (2008) considered the scatter search as a method for solving MKP01.

3.4 Methodology

In order to test the proposed approaches, standard test-data must be used. For this, several experimental data-sets have been created, some are simple and others are complex. The community of researchers in this subject uses the same data to validate their approaches and to make comparisons. All these data are gathered and published in a OR-Library. Three data-sets are used to conduct the experiments: the SAC-94, Chu&Beasley and Glover&Kochenberger.

- SAC-94:This data-set is composed of 54 instances where the number of items n varies from 2 to 30 and the number of dimensions m varies from 10 to 105. This data-set is divided into six groups of MKP01 problems, all considered by many researchers as simple and the optimal solution of each instance is known.

Table 3.1: SAC-94 description

Group	Number of instance	Dimensions (N, M)
hp	2	(28,4), (35,4)
pb	6	(27,4), (34,4), (29,2), (20,10), (40,30), (37,30)
pet	6	(10,10), (15,10), (20,10), (28,10),(39,5), (50,5)
wento	2	(60,30), (60,30)
weing	8	(28,2), (28,2), (28,2), (28,2), (28,2), (28,2), (105,2), (105,2)
weish	30	(30,5), (30,5), (30,5), (30,5), (30,5), (40,5), (40,5), (40,5), (40,5), (50,5), (50,5), (50,5), (50,5), (60,5), (60,5), (60,5), (60,5), (70,5), (70, 5),(70,5), (70,5), (80,5), (80,5), (80,5), (80,5), (90,5), (90,5), (90,5), (90,5), (90,5)

Table 3.1 contains the description of the used data. Here N and M represent the number of items and the number of constraints (the number of dimensions) respectively. Column Ins represents the number of instance in each group of data.

- Chu&Beasley: These data are more complex and larger than SAC-94. They are composed of 270 instances classified in nine groups. The number of items n= 5, 10, 30 and the number of dimensions m= 100, 250, 500. These data have been generated in 1998 based on a complex mathematical formula. The major part of the instances has been solved to the optimality, but for some instances the optimal solution is still unknown.

- Glover&Kochenberger: This data-set consists of 11 larger and more complex instances. The number of items n varies from 15 to 50 and the number of dimension M varies from 100 to 1500. The optimal solution for all the instances is still unknown.

This work intends to compare the proposed approaches with exact methods proposed in the CPLEX. CPLEX is a software developed by IBM that was developed to solve complex linear programs. Several exact methods of optimisation are implemented in CPLEX such as: Branch and Bound, Branch and Cut, Simplex. In our project we compare the results obtained by our approach with those obtained by the CPLEX.

3.5 Conclusion

The study of this chapter allows us to reach the following conclusions: firstly, in terms of quality, the exact methods exceed the approximate methods, however, they required more time, so their choice depends on the data size; secondly, although some heuristics are effective, the problem of the local optimum requires caution when using them; thirdly, the hybridisation may be a good option to avoid the local optimum and have a good quality in an acceptable time. Here, hybridisation is either between several approximate methods or between exact and approximate methods; finally, the exploitation of prior knowledge on data could increase the solution quality and accelerates the convergence to the optimal solution.

CHAPTER 4

LOCAL-SEARCH METHODS FOR MKP01 AND MU-WDP

4.1 Introduction

One of the objectives of our work, is to investigate the different kind of solution methods. Consequently, it seems logical to start with the simplest ones: the local-search meta-heuristics. Two contributions are suggested in this chapter. The first is a new local-search algorithm resulting from the combination of Simulated Annealing (SA) and Stochastic Local Search (SLS). The algorithm named SLSA exploits the advantages of both algorithms and proposes a modified search and repair structures. Moreover, SLSA is examined on SAC-94 and Glover & Kochenberger data-sets. The second attempts to investigate the application of SLS, TS and Hill-Climbing (HC) to solve the Winner Determination Problem in Multi-Unit Combinatorial Auctions (the MU-WDP), which is a particular case of the MKP01.

This chapter is organised into two main sections, Section 4.2 details the first proposed approach, which is SLSA method for the MKP01. Section 4.3 describes the second approach, a local search methods for the MU-WDP. The main conclusions of both approaches as well as some perspectives are summarised in Section 4.4.

4.2 Local search method for the MKP01

Generally, it is known that the local-search meta-heuristics converge quickly but may converge to a local optimum. Simulated Annealing (SA) (explained in Section 3.3.2.1) is a popular local-search heuristics owing to its simplicity and effectiveness. Similarly Stochastic Local Search (SLS) is an efficient local-search meta-heuristic as well. SA is able to find zones not yet visited thanks to its neighbourhood creation strategy. Nevertheless, its searching process may visit the same solution several times. Comparatively, SLS (H. H. Hoos & Boutilier, 2000; H. Hoos & Stutzle, n.d.) is effective in terms of diversification capacity, due to its neighbourhood creation strategy. But this strategy, based on a random and HC techniques requires a long time to reach good results. The Stochastic Local search-Simulated Annealing algorithm (SLSA) to solve MKP01 is proposed here. SLSA is a hybridisation between SLS and SA. Three main modifications are proposed in SLSA. The proposed SLSA approach is compared with SA and SLS using data-sets of OR-Library [1].

4.2.1 Stochastic Local Search (SLS)

SLS is a local-search meta-heuristic that performs a certain number of local steps that combines diversification and intensification strategies to locate a high-quality solution. The intensification phase consists in selecting a best neighbour solution. The diversification phase consists in selecting a random neighbour solution. The diversification phase is applied with a fixed probability $wp > 0$ whereas the

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/

intensification phase with a probability 1 - wp. SLS starts with an initial solution X generated randomly according to the RK encoding. Then, it performs a certain number of local steps as follows:

- *Step 1.* Creating neighbour solution X' to X by selecting an item I to be added in the current solution X. At each step, the item to be accepted is selected according to one of the two following criteria:
  - Choose one item in a random way with a fixed probability $wp > 0$.
  - Choose the best item to be accepted.
- *Step 2.* To make X' a feasible solution, it is repaired by removing an item repeatedly. The item to be deleted is chosen in two manners according to the probability P. either an item is randomly chosen or the worst one in X' is found and removed.
- *Step 3.* Save in X^* the best neighbour feasible solution found so far.

The process is repeated for a certain Number of Iterations (NI) that was determined empirically. The SLS algorithm could be summarised by the pseudo-code of Algorithm 1.

4.2.2 The proposed SLSA

We propose the combination of the SLS and the SA to produce a new approach called SLSA. In the following, we explain the main component of the proposed approach. The SLSA process is based on two phases as follows:

4.2.2.1 Create the initial solution

The SLSA begins by creating an initial feasible solution. For that, the Random Key method (Bean, 1994) is used. n values in [0, 1] are generated randomly and arranged in ascending order, such that each item corresponds to one of the

Algorithm 1 the SLS method.

```
Require: a feasible solution $X$, $maxiter$, $wp$
Ensure: a better feasible solution $X^*$
  for $i = 1$ to $maxiter$ do
    generate r, r $\in [0,1]$ ;
    if r < $WP$ then
      $x_i = x_{rand}$ (*Step 1)
    else
      $x_i = x_{max}$ (*Step 2)
    end if
    $X = X \cup \{x_i\}$; Update (f(X) and somConflit);
    while there is conflict do
      $x_i = x_{min}$; $X = X - \{x_m\}$; Update (f(X) and somConflit);
    end while
    if $(F(X) > F(X^*))$ then
      $X^* = X$;
    end if
  end for

  return the best solution $X^*$.
```

generated values. Secondly, the solution is built by adding items one after one in order, as long as all constraints are satisfied. If the addition of an item leads to break at least one constraint then it is ignored. This operation continues until the last item. Thirdly, the objective function of the solution is calculated. The creation of feasible solution by the RK is the first phase in SLSA. It is followed by the optimisation phase.

4.2.2.2 Optimisation by SLSA

Here the initial feasible solution X is iteratively modified. The SLSA process performs a certain number of local steps that consists in: Create a neighbour solution X, Repair the created neighbour solution, Record the best solution and update the Temperature.

- *Step 1. Create a neighbour solution X' of X.* At each operation and with the probability $wp \in [0,1]$, the item accepted to be packed is selected according to one of the two following criteria:

  - *SA strategy.* One item I is chosen arbitrary. If I increases the objective function $f(X)$ then it will be packed to the knapsack, otherwise it will be accepted if the exponential $\exp^{(\Delta f()/T)} > r$ where r is a probability generated randomly, $\Delta f() = f(X') - f(X)$ and T is the temperature value initially equal to T_0 relatively high.
  - *Mutation of an item.* replace an item in X' by another not in X'. The replaced and the replacement items are chosen randomly.

- *Step 2. Repair the solution.* The first step may cause a conflict. To eliminate all conflicts, and make X' a feasible solution, it is repaired by removing an item. The item to be deleted is chosen in two manners according to the probability P. either an item is randomly chosen or the worst one in X' is found and removed.

- *Step 3.* If the created neighbour solution outperforms the objective function value ($f(X') > f(X^*)$) then it is recorded as the best solution found so far.

- *Step 4.* The temperature value is updated. In our case the decreasing rule is found empirically as $T = T - 0.0105$.

The optimisation phase is repeated for a certain Number of Iterations NI, which was determined empirically. The SLSA algorithm is sketched in Algorithm 2.

4.2.3 Simulation results

The algorithms SA, SLS and SLSA are coded using C++ and compiled on a PC having 2 GHz Intel Core 2 Duo processor and 2 GB RAM. In order to evaluate the efficiency and the performance of the proposed SLSA, it was tested on the SAC-94 standard test problems (divided into six different sets) which are available

Algorithm 2 The SLSA pseudo-code.

```
Require: a feasible solution $X, NI, wp, T_0$
Ensure: a better feasible solution $X^*$
 1: for $Cpt = 1$ to $NI$ do
 2:     if $(r < wp)$ then
 3:         $I_1 = RandItem\ ()$; $I_1 \in X'$
 4:         if $(f(X') + p_{I_1} > f(X))$ then
 5:             $X' = X' \cup \{I_1\}$
 6:         else
 7:             if $(r_1 < \exp^{(\Delta f/T)})$ then
 8:                 $X' = X' \cup \{I_1\}$
 9:             end if
10:         end if
11:     else
12:         $I_2 = RandItem()$; $I_2 \in X'$
13:         $I_3 = RandItem()$; $I_3 \notin X'$
14:         $X' = X' - \{I_2\}$
15:         $X' = X' \cup \{I_3\}$
16:     end if
17:     while $(ExistConflict\ (X'))$ do
18:         if $(r_2 < P\ )$ then
19:             $I_{min}$ =$WorstItem()$; $I_{min} \in X'$
20:             $X' = X' - \{I_{min}\}$
21:         else
22:             $I_4 = RandItem()$;$I_4 \in X'$
23:             $X' = X' - \{I_4\}$
24:         end if
25:     end while
26:     if $(f(X') > f(X^*))$ then
27:         $X^* = X'$
28:     end if
29:     $T = T - CT$
30: end for
31: Return the best solution $X^*$.
32: Where $r, r_1, r_2 \in [0,1]$. $T$ is the temperature. $\Delta f = f(X^*) - f(X')$, $p_{I_1}$ is the
    value of item $I_1$ and $CT$: coefficient of temperature update.
```

at the OR-Library [2]. These data-sets are a real-world problems widely used to test and validate the algorithms effectiveness in the optimisation community.

[2]http://www.brunel.ac.uk/~mastjjb/jeb/info.html

Table 4.1: The parameters of algorithms.

Parameter	Description	Value
NI	Number of iteration of SLSA	100000
wp	Probability of SLSA	0.98
T_0	Initial temperature	50
CT	Coefficient T update	0.0105
Run	Number of run times	30

These problems dimensions vary as $m = 2$ to 30 and $n = 6$ to 105. After several experiments, we set the parameters for SLSA as in table 4.1.

Table 4.2 shows the obtained results of the application of SA, SLS and SLSA algorithms on the 54 instances. Column $A.V.F$ means the average fitness of all the 30 runs while column $D.F.O$ represents the deviation from the optimal.

From these results we have identified various observations. We observed that the $A.V.F$ obtained by SLSA is better than SA and SLS in all instances of the groups: *pet, sento and weish*, furthermore in 43 of the 54 instances $A.V.F$ SLSA is better. We see that instances *pet1, pet2, weing2* and *weing3* are the less complex ones; it is the reason why the $A.V.F$ obtained by SA, SLS and SLSA is equal to the optimal. But only SLSA reached the optimal solution in all the 30 runs in the two instances *weish1* and *weish4*. In all groups, in terms of average $D.F.O$, SLSA surpassed SA which surpassed SLS. Also we found that in terms of average $D.F.O$, SLSA outperformed SA of 1.2% SLS of 1.31%. Additionally, in terms of average $D.F.O$, SA was slightly better than SLS of 0.11%. Fig. 4.1 shows clearly the difference.

In this study, three major modifications have been made to the algorithms. One of them concerned all the algorithms when the two others characterised only the SLSA. These modifications gave to the algorithms more effectiveness. Firstly, introducing the SA neighbourhood creation mechanism in SLSA allowed to increase the quality of the obtained results. Secondly, the mutation represents the change that has the most significant impact on SLSA. Finally, repairing the solution by removing an item chosen randomly with probability P performed the

Table 4.2: SLSA vs. SA and SLS

	SA		SLS		SLSA	
Group	A.V.F	D.F.O	A.V.F	D.F.O	A.V.F	D.F.O
hp	**3347,97**	97,95	3345,7	97,88	3345,67	97,88
	2998,07	94,10	2993,93	93,97	**3011,27**	94,52
pb	3018,67	97,69	**3027,9**	97,99	3027,07	97,96
	3034,2	95,24	3034,27	95,24	**3031,87**	95,16
	87063,5	91,48	86672,3	91,07	**90793,7**	95,40
	2095,8	97,98	**2101,97**	98,27	2098,6	98,11
	679,2	87,53	679,233	87,53	**727,033**	93,69
	935,5	90,39	928,333	89,69	**999,333**	96,55
pet	87061	100	87061	100	87061	100
	4015	100	4015	100	4015	100
	6120	100	6120	100	6120	100
	12206,7	98,44	12200	98,39	**12285,3**	99,08
	10384,4	97,80	10373,5	97,70	**10394,6**	97,90
	15925,3	96,30	15938,2	96,38	**15977,7**	96,62
sento	7675	98,75	7675	98,75	**7690,57**	98,95
	8580,8	98,38	8587,43	98,46	**8670,93**	99,41
weing	138453	98,00	138871	98,30	**141267**	99,99
	130883	100	130883	100	130883	100
	95677	100	95677	100	95677	100
	115709	96,96	114896	96,28	**118598**	99,38
	96936,8	98,12	96897,5	98,08	**98693,5**	99,90
	130610	99,99	130610	99,99	130610	99,99
	1087448	99,27	1086462	99,18	1086790	99,21
	583048	93,39	575396	92,16	576703	92,37
weish	4491,33	98,62	4505,2	98,92	**4554**	100
	4531,17	99,89	4531,5	99,90	**4535,17**	99,98
	3993,2	97,04	4002,67	97,27	**4090,77**	99,41
	4512,47	98,93	4516,17	99,01	**4561**	100
	4384,33	97,12	4391,97	97,29	**4512,2**	99,96
	5327,13	95,86	5304,6	95,45	**5465,33**	98,35
	5326,1	95,67	5312,03	95,42	**5470,27**	98,26
	5326,07	95,02	5318,03	94,88	**5483,03**	97,82
	5218,8	99,48	**5221,07**	99,52	5218,8	99,48
	6166,23	97,27	6166,33	97,27	**6224,13**	98,18
	5059,87	89,66	4978,87	88,23	**5363,67**	95,04
	6227,13	98,23	6217,57	98,08	6211,7	97,99
	5902,17	95,83	5903,8	95,85	**5977**	97,04
	6769	97,33	6765,37	97,28	6743,57	96,97
	7199,77	96,17	7208,23	96,28	**7336,9**	98,00
	7053,73	96,773	7051,53	96,74	**7124,87**	97,74
	8503,43	98,49	8504,53	98,51	**8507,53**	98,54
	9249,5	96,55	9245,53	96,50	**9291,2**	96,98
	6952,73	90,31	6921,7	89,91	**7207,03**	93,62
	9121,4	96,52	9155,83	96,88	**9282,5**	98,22
	8838,43	97,40	8842,53	97,44	**8860,93**	97,65
	8246,73	92,17	8178,83	91,41	**8417**	94,07
	7635,63	91,51	7611,1	91,21	**7722,83**	92,55
	9730,67	95,21	9729,57	95,20	**9777,17**	95,66
	9589,67	96,48	9595,13	96,54	**9710,37**	97,69
	8733,7	91,12	8715,93	90,94	**8944**	93,32
	8888,67	90,52	8873,4	90,36	**9145,17**	93,13
	8996,63	94,78	8958,67	94,38	8923	94,00
	8866,93	94,22	8847,47	94,02	8810,03	93,62
	10681,3	95,44	10676,9	95,40	**10830,9**	96,78
Total Average	52693,1	96,24	52512,9	96,13	**52829,1**	**97,44**

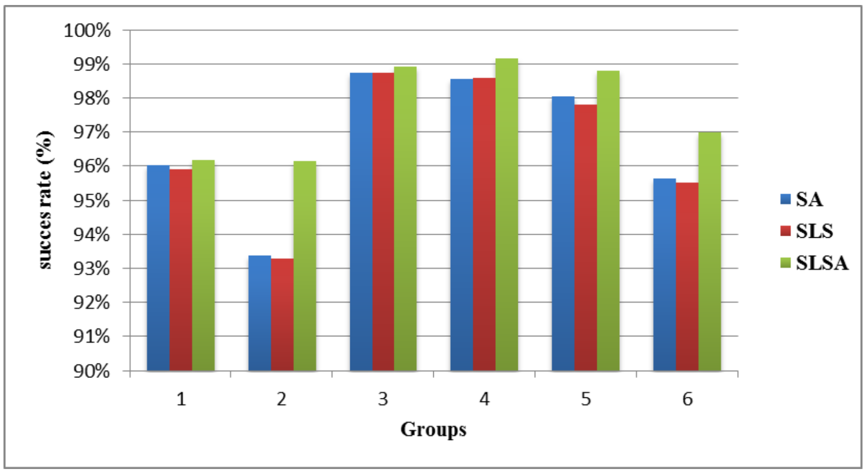

Figure 4.1: SLSA vs. SA and SLS.

obtained results by SA, SLS and SLSA. This mechanism prevents the search process from reproducing solutions already visited. It provides effective means to conduct the search process in zones not yet discovered. Thanks to this mechanism and the SA neighbourhood strategy SLSA avoids stagnation in the local optima and succeed to find solution very close to the optimal.

4.3 Local search methods for the MU-WDP

As a second part of this chapter, the study adapts some local-search methods to deal with the MU-WDP, which is a particular and applicative case of the MKP01. The MU-WDP is an NP-hard combinatorial optimisation problem (Sandholm et al., 2002). Several works tried to study the similarity between the MKP01 and the MU-WDP. Thus, it has been demonstrated that the MU-WDP can be modelled as a MKP01 (Holte, 2001; Kelly, 2004). In this section, we propose three local search methods to solve the MU-WDP. Therefore, we present Hill-Climbing (HC), Tabu Search (TS) and Stochastic Local Search (SLS) for the MU-WDP.

4.3.1 Background

A background about the main shared components of the three approaches is presented first. It concerns the solution representation, the Random Key encoding strategy used to generate the initial solution and the objective function used to measure the quality of solutions.

4.3.1.1 Solution representation

A solution for the MU-WDP is an allocation S which can be represented by a vector with a variable length. Each of those components S_i represents the winning bid number.

4.3.1.2 Generate the initial solution

The initial solution is generated at random using the RK strategy (see Section 4.2.2.1). RK encoding mechanism permits to generate and manipulate only feasible solutions. The initial solution is created as follows:

- *Step 1.* Generate n real numbers sequenced by an r order, where n is the number of bids and the r order is a permutation of key values.
- *Step 2.* Select the bid having the highest order value to include in the allocation, knowing that the allocation is initially empty. Then, the bid having the second-highest order value is accepted if its acceptance with accepted bid currently in the allocation verifies the constraint that means that for any good i, the sum of units of i over all the winning bids in the current allocation does not exceed mu_i, otherwise it is discarded. This process continues until having examined the n bids. The output is a subset of bids that can be a feasible solution to the MU-WDP.

4.3.1.3 Objective function

The quality of a solution S is evaluated by calculating its objective function. The objective function of an allocation is equal to the sum of prices of the winning bids $S = \{B_1, B_2, \ldots, B_l\}$. Where l is the number of element of the allocation S.

$$f(S) = \sum_{i=1}^{l} p_i x_i \tag{4.1}$$

4.3.2 The proposed local search algorithms

In this section, the aim is to adapt the Hill-Climbing (HC), Tabu Search (TS) and Stochastic Local Search (SLS) methods for solving the MU-WDP.

4.3.2.1 Hill-Climbing (HC)

The Hill-Climbing (HC) method is an iterative improvement method that consists of two essential components: finding a neighbour solution and a exploring the neighbourhood solutions to find good ones (Hansen & Mladenović, 1999).

More precisely, it consists of the following steps:

- *Step 1.* start with a random initial configuration or an arbitrary solution S_0.
- *Step 2.* consider this configuration as a solution with the problem $S = S_0$, measure the quality of S by $F(S)$ where F is the objective function to be optimised.
- *Step 3.* choose a neighbour solution $S^{'}$ of S such as $F(S^{'}) > F(S)$, replace S by $S^{'}$.
- *Step 4.* Repeat *Step 3.* until for all neighbour $S^{'}$of S, $F(S^{'}) \leq F(S)$.

4.3.2.2 Tabu Search (TS)

The proposed TS for the winner determination problem starts from an initial allocation created using the Random Key RK method. Then, it tries to locate for the best allocation by generating neighbour allocation. This is done by applying the three steps:

- *Step 1. Generate a neighbour allocation from the current allocation.* A bid O is chosen such as it does not belong to the *Tabu List* and its price is the highest. When O is found, it will be included into the allocation, and added into the *Tabu List*. After that, we update the allocation revenue and a vector somConflit allows controlling the conflicts.
- *Step 2. Conflict elimination.* To do that, the bid O_m having the minimal price is found and deleted from the allocation. When a bid is deleted from the solution, its price and goods quantities requested must be omitted. This procedure is repeated while the conflict persists.
- *Step 3. Saving the best allocation.* For each neighbour allocation, the revenue is evaluated, so if it is more than the revenue of the best allocation then the neighbour allocation becomes the best allocation S^*.

The search process is repeated for a number of iterations fixed empirically. At the end, we obtain the best solution that gives the optimal revenue to the sellers. The TS algorithm is sketched in Algorithm 3.

Algorithm 3 TS pseudo-code for the MU-WDP.

```
Require: a MU-WDP instance, $maxiter$, a collection $S$,
Ensure: An improved allocation $S$
 1: The Tabu List, is initially empty: Tabu List = $\phi$;
 2: Generate an arbitrary feasible allocation $S$ using RK encoding;
 3: for $iter = 0$ to $maxiter$ do
 4:     Choose $O$ from reparation bids having the maximal price and $O \cap$ Tabu List
        = $\phi$;
 5:     $S = S \cup O$;
 6:     Update (Revenue and somConflit);
 7:     Tabu List = Tabu List $\cup$ $O$;
 8:     while there is conflict do
 9:         $O_m$ in $S$ having the minimal price;
10:         $S = S - O_m$;
11:         Update (Revenue and somConflit);
12:     end while
13:     if $F(S) > F(S^*)$ then
14:         $S^* = S$;
15:     end if
16: end for
17: return the best solution $S^*$.
```

4.3.2.3 Stochastic Local Search (SLS)

The SLS (see Section 4.2.1) has been applied successfully on several optimisation problem such as the single-unit winner determination problem in combinatorial auctions (Boughaci et al., 2010; Boughaci, 2013). In this section, we propose an SLS adaptation for the multi-unit winner determination problem (the MU-WDP). The proposed SLS method starts with an initial allocation S generated randomly according to the RK encoding. Then, it performs a certain number of local steps that consists in selecting a O to be added in the current allocation S and in removing all conflicting bids that occur in the current allocation.

At each step, the bid to be accepted is selected according to one of the two following criteria:

1. The first criterion ($step1$ of Algorithm 4) consists of choosing the bid in a random way with a fixed probability $wp > 0$.

2. The second criterion ($step2$) consists of choosing the best bid (the one maximising the auctioneer's revenue when it is selected) to be accepted.

The process is repeated for a certain number of iterations $maxiter$, which is determined empirically.

The SLS algorithm is sketched in Algorithm 4.

Algorithm 4 : The SLS method.

```
Require: a MU-WDP instance, $maxiter$, $wp$
Ensure: an allocation $S$
 1: Generate an initial solution according to RK $S$
 2: for $I = 1$ to $maxiter$ do
 3:     r ⇐ random number between 0 and 1;
 4:     if r < $wp$ then
 5:         $O$ = pick a random bid (*Step 1)
 6:     else
 7:         $O$ = pick a bid having the maximum price; (*Step 2)
 8:     end if
 9:     $S = S \cup O$;
10:     Update (Revenue and somConflit);
11:     while there is conflict do
12:         $O_m$ = the bid having the minimum price;
13:         $S = S - O_m$;
14:         Update (Revenue and somConflit);
15:     end while
16:     if ($F(S) > F(S^*)$) then
17:         $S^* = S$;
18:     end if
19: end for
20: return the best allocation $S^*$.
```

4.3.3 Simulation results

In order to evaluate the performance of the three proposed approaches HC, TS and SLS in solving the MU-WDP, they have been implemented and evaluated on some benchmarks of the problem. The algorithms have been run on a hp Compaq 610 Laptop with a core 2 duo 2 GHz CPU and 2 GB of memory.

Table 4.3: The Data set description.

data	bid	good	good/bid	price	quantity
B1	1400	1500	[0, 110]	[322, 325]	[900, 920]
B2	1300	900	[0, 200]	[600, 605]	[1200, 1230]
B3	800	800	[0, 400]	[400, 420]	[800, 830]
B4	900	900	[0, 250]	[200, 210]	[1000, 1100]
B5	700	1000	[0, 60]	[100, 105]	[200, 220]
B6	1000	600	[0, 55]	[150, 155]	[100, 115]
B7	1100	1500	[0, 70]	[700, 710]	[200, 220]
B8	1500	1500	[0, 90]	[800, 820]	[600, 630]
B9	1500	1000	[0, 300]	[10000, 10500]	[3000, 3100]
B10	1200	1000	[0, 120]	[500, 510]	[900, 950]

4.3.3.1 Data issue

Despite the MU-WDP has been widely investigated, to our knowledge, no standard test data has been created. Additionally, many authors have generated their own data to validate their approaches. Test data-set used in this study is randomly generated. Ten benchmarks of various sizes have been generated.

Table 4.3 gives information about the considered benchmarks. The first column gives the name of benchmark, column *bid* the number of bids, column *good* the number of goods, column *good/bid* the number of goods per bid, column *price* the interval values of bids and column *quantity* the available quantities of goods. The benchmarks were generated so as to increase the complexity of the problem by creating much competition between the variables of the problem (the bids). we notice that the prices of bids are very brought closer as well as the available quantity of each good.

4.3.3.2 Parameters tuning

The adjustment of the different parameters of the proposed approaches SLS, TS and HC is fixed by an experimental study.

- *The TS parameters:* are the maximum number of iterations is fixed to 5000 iterations. This number of iteration is sufficient and allows the process to find the best solution. The size of the *Tabu List* is set to 40 elements.

Table 4.4: Numerical results of SLS, TS, HC and CPLEX.

	SLS		TS		LS		CPLEX
data	**Time** (s)	**Rev** (Mu)	**Time** (s)	**Rev** (Mu)	**Time** (s)	**Rev** (Mu)	**Rev** (Mu)
B1	0.047	**3573.9**	3.222	3573.9	0.002	3569.5	3561.5
B2	1.233	**4839.2**	2.265	4821.1	0.001	4833.6	4832
B3	0.078	**839.8**	4.84	839.8	0.0001	839.1	839.7
B4	3.074	**839.6**	2.646	839.6	0.001	837.9	838.7
B5	0.936	**419.6**	2.374	419.4	0.0008	417.2	419.0
B6	0.514	**309.8**	2.73	309.7	0.0004	308.8	309.8
B7	1.108	**2129.7**	2.488	2129.7	0.0001	2128.8	2128.4
B8	1.2	**7378.4**	3.877	7303.8	0.0032	7035.6	7351.4
B9	0.062	**6053.9**	3.568	5789.6	0.0004	5047.2	6042.7
B10	2.075	**5600.2**	3.892	5098.7	0.0024	5283.3	5597.5

(s) : second and (Mu) : Monetary unit.

- *The SLS parameters:* are the maximum number of iterations is fixed to 50000 iterations. The value of probability $wp = 0.87$.

- *The HC parameters:* are the searching process is stopped according to the increasing of the objective function F, when F is not increasing any more and no bid remains then the process stops.

4.3.3.3 Numerical results

Table 4.4 gives the results obtained by SLS algorithm, HC and TS for the considered the MU-WDP benchmarks. For each method, we give the CPU time (*Time* in $second$) and the revenue of the best solution found by each approach (*Rev* is in $Monetary\ unit$).

In order to show the performance of the SLS approach, we conducted a comparison with the CPLEX solver 12.5. We note that for each benchmark, the CPLEX is launched for fifteen minutes (15 min).

Table 4.4 gives the average results obtained after multiple runs of the algorithms where Rev is the average revenue and $Time$ is the average CPU consumed in second.

According to the obtained results we released the following remarks:

- for all the benchmarks we noticed that the approach SLS is always able to give the same results of revenue after several runs. The TS always gives the same results of revenue for the benchmarks 1, 3, 4 and 7, whereas HC finds various results for each run for all benchmarks.
- according to the numerical results, SLS succeeds to find solutions of good quality for the all checked benchmarks. For benchmarks 1 and 3 the quality solutions found by SLS is similar to those found by TS. We remark that, the Benchmark 6 is not very complex since CPLEX gave the same revenue as SLS. However, the simple HC fails to obtain good results for all the benchmarks compared to both SLS and TS.
- In term of CPU time point of view, HC is faster than both SLS and TS, but the disadvantage is that it falls quickly into the local optimum.

4.4 Discussion & conclusion

This chapter provided two main solutions for the MKP01 and the MU-WDP. Therefore, the work has been divided into two parts.

The first part proposed a local search solution to the 0-1 multidimensional knapsack problem (MKP). The suggested solution is the combination of the Stochastic Local Search (SLS) with the simulated annealing (SA). The proposed approach is called the Stochastic Local Search-Simulated Annealing (SLSA). Several tests and comparison were carried out on a large range of benchmarks known by their complexity. In conclusion, the use of the three techniques allowed SLSA to obtain good results and surpass significantly SA and SLS. Similarly SLSA succeed to reach or at least be close to the optimal, indeed the overall success rate is of 97.44%.

The second part is an extension of the first one. It proposed three local search methods for the MU-WDP. The three proposed approaches SLS, Tabu Search (TS) and Hill-Climbing (HC) have been implemented and tested on some benchmarks generated randomly. Also, A comparison with the CPLEX solver 12.5 has been conducted. According to the experimental study, SLS outperforms the other methods with encouraging results. The comparative study with CPLEX confirms the conclusions.

CHAPTER 5

HYBRID META-HEURISTICS FOR THE MKP01

5.1 Introduction

Meta-heuristics may be a good methods for solving complex CO problem. However, there are several other CO problems where they fail, hybridisation is therefore encouraged. Hereby, three hybrid meta-heuristic approaches are proposed for the MKP01. All the proposed approaches combine a local-search algorithm with a population-based algorithm. Furthermore, the population-based algorithm provides good exploration of the solution space, while the local-search improves the quality of the solutions by exploiting deeply their neighbourhood.

The first approach proposed here, is based on GA as population-based algorithm supported by SA and SLS as local-search algorithms. Thus, two methods are suggested GA-SA which combines GA with SA and GA-SLS which combines GA with SLS. Both methods are examined using the well-known test data-sets and compared to other methods of the literature.

The second approach extends the first one and proposes the Memetic Search

Algorithm (MSA). MSA consists on the hybridisation of a modified GA and SLSA (explained in Section 4.2). Several experimental tests are carried out to examine the effectiveness of MSA on the MKP01.

The last approach hybridises a new version of HS denoted Self-Adaptive Harmony Search (SAHS) with SLS. The resulting method denoted SAHS-SLS introduces many strategies to dynamically calculate the HS as well as the SLS parameters.

5.2 Background

Three local-search methods are used to build the proposed approaches, all have already been described in previous chapters (see Section 3.3.2.1, 4.2.1 and 4.2.2 for the SA, SLS and SLSA descriptions respectively). The main parts of each algorithm are maintained unchanged. Specifically, in our approaches, local-search is applied on a solution chosen from the population of the population-based method. It is executed for a number of iterations and returns a better neighbour solution - if one is found - or the same solution otherwise.

Moreover, two population-based methods are combined with the above mentioned local-search methods, which are GA and HS, these methods have been described in Section 3.3.2.2.

5.3 GA-SA and GA-SLS for the MKP01

This section describes GA-SA and GA-SLS. SA and SLS have already been described in detail in Section 3.3.2.1 and 4.2.1 respectively. Only GA-SLS is discussed here as its structure is similar to GA-SA. The difference between both methods is only on which local-search algorithm is applied (i.e. SA or SLS).

5.3.1 The GA in GA-SLS and GA-SA

GA-SLS (resp. GA-SA) starts by executing the GA process. The GA steps and some modifications concerning selection, crossover, mutation and replacement operations are presented in following. Fig. 5.1 summarises the GA steps.

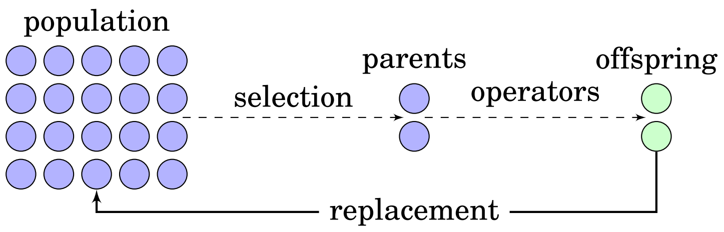

Figure 5.1: The scheme of the Genetic Algorithm process.

1. *Population initialisation*. The GA in GA-SLS (resp. GA-SA) initialises a population P of size $PS > 1$ individuals. The RK method is used to create the individuals. To create an individual, n values in the range [0, 1] are generated randomly and arranged in an ascending order, such that each item corresponds to one of the generated values. The individual is built by adding items one after one, according to the order, as long as all constraints are satisfied. The creation of individuals by the random key is repeated until filling the entire population.

2. *Selection*. The selection operator uses the fitness to choose the parents. The individuals with higher values are most likely to be selected. Several selection approaches exist such as elitist, roulette or random, etc. In GA-SLS (resp. GA-SA), the two individuals X_1,X_2 with the first and the second best fitness values are designated as parents for the genetic operations.

3. *Crossover*. X_1,X_2, are used to make the crossover. GA-SLS (resp. GA-SA) uses the random multiple point crossover method. The number of crossing points ($N.C.P$) is calculated by the following formula:

$$N.C.P = \frac{1}{k} Min(|X_1|, |X_2|) \tag{5.1}$$

Where k is a positive integer such that $k < n$.

A queue Q is used to store the list of individuals that have already participated in a crossover. The purpose of this list is to prevent parents to be selected more than once during a number of iterations. The size of the Q is defined according to the population size PS so that it turns the major part of its individuals $WT = F * PS$, with $F < 1/2$ is a coefficient to calculate the size of Q.

4. *Offspring repair*. The offspring X'_1, X'_2 may be unfeasible solutions. The process to repair the offspring can be described by the following steps:

Algorithm 5 Offspring repair process.

```
1: while (ConflictExist($X$)) do
2:     $I_{worst} = WorstItem()$; $I_{worst} \in X$.
3:     $X = X - \{I_{worst}\}$.
4: end while
```

5. *Fitness function.* Offspring X'_1 and X'_2 represent a feasible solutions, these lead to calculate their fitness $f(X'_1)$ and $f(X'_2)$. The fitness of an individual is calculated by the sum of the profits of the items that compose it.

6. *Mutation.* A number of items (MNI) from offspring X'_1 and X'_2 are replaced by others items selected randomly from the best individual found so far X_{best}. The replaced items must not be included in the concerned offspring. The procedure can be summarised by the following pseudo code:

7. *Replacement and stopping criteria.* X'_1,X'_2 replace the two worst individuals in the population. The optimisation process is repeated until the stopping criterion is checked. The criterion for stopping the optimisation process is a

Algorithm 6 Mutation process.

```
1: for $Cpt = 1$ to $MNI$ do
2:     $I_x = RandItem()$; $I_x \in X$
3:     $I_(xb) = RandItem()$; $I_{xb} \in X_{best}$, $I_{xb} \notin X$ and $I_x \neq I_{xb}$
4:     $X = X - \{I_x\}$
5:     $X = X \cup \{I_{xb}\}$
6: end for
```

limited number of iteration NI determined empirically according to the size of the studied problem.

5.3.2 The proposed GA-SLS and GA-SA

The GA-SLS procedure consists in:

- *Step 1.* Create the initial population P using the RK method (see Section 4.2.2.1) and initialise $Q = \{\}$, NI and $T = T_0$ (T for GA-SA only).
- *Step 2.* Select two parents X_1 and X_2 that are the two best individuals in P and $X_1, X_2 \notin Q$.
- *Step 3.* Exchange NCP items between the parents X_1 and X_2 to produce two new infeasible offspring X'_1 and X'_2, then if conflict exists in X'_1 or X'_2, repeatedly remove either the worst items or an item chosen randomly according to a probability rp.
- *Step 4.* Push the two parents X_1 and X_2 in Q. Apply the local-search (SLS for GA-SLS or SA for GA-SA) on offspring X'_1, X'_2. Find the best individuals X_{best} in P and replace randomly a number of items in X'_1 and X'_2 by items in X_{best}.
- *Step 5.* If the quality of X'_1 and X'_2 is better than the two worst individuals in P, then they replace them. If the number of iterations NI is not attend then go to *Step 2.*. Otherwise return the best individual in P.

The GA-SLS and GA-SA can be expressed by Algorithm 7.

Algorithm 7 GA-SLS Algorithm.

```
Require: An MKP01 instance, $NI$ and $Q = \phi$.
Ensure: An best solution found $X^*$.
 1: Create the initial population $P$ by the RK method.
 2: for ($Cpt = 1$ to $NI$) do
 3:     Selection of the two best individuals $X_1, X_2$ in $P$ and $X_1, X_2 \not\in Q$.
 4:     Crossover $X_1, X_2$ to produce offsprings $X'_1, X'_2$
 5:     Repair offsprings $X'_1, X'_2$
 6:     Apply the local-search method on $X'_1, X'_2$
 7:     Mutation on $X'_1, X'_2$ with $X_{best}$ of $P$
 8:     $X_{worst} \longleftarrow$ the worst individual in $P$
 9:     if ($f(X'_1) > f(X_{worst})$) then
10:         $P = P - \{X_{worst}\}$
11:         $P = P \cup \{X'_1\}$
12:     end if
13:     $X_{worst} \longleftarrow$ the worst individual in $P$
14:     if ($f(X'_2) > f(X_{worst})$) then
15:         $P = P - \{X_{worst}\}$
16:         $P = P \cup \{X'_2\}$
17:     end if
18:     $Q = Q \cup \{X_1, X_2\}$
19: end for
20: Return the best individual found.
```

5.3.3 Simulation results

GA, GA-SA and GA-SLS were implemented in C++ on 2 GHz Intel Core 2 Duo processor and 2 GB RAM. They were tested on the OR-Library [1] 54 benchmarks, with $m = 2$ to 30 and $n = 6$ to $n = 105$ and on the OR-Library GK with $m = 15$ to $m = 50$ and $n = 100$ to $n = 1500$. In all experiments the parameters are chosen empirically such as: the number of iteration $NI = 30000$, the population size $PS = 100$, the waiting time $WT = 50$, the number of crossing bites $NCB = 1/10$. the initial temperature $T_0 = 50$, the walk probability $wp = 0.93$, the number of local iteration $N = 100$ and the number of runs is 30.

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

Table 5.1: Comparison of GA, GA-SA and GA-SLS on SAC-94 datasets.

		GA		GA-SA		GA-SLS	
Dataset	Opt	Result	GAP	Result	GAP	Result	GAP
hp	3418	3381,07	1,080	3418	0	3418	0
	3186	3120,63	2,052	3186	0	3186	0
pb	3090	3060,27	0,962	3090	0	3090	0
	3186	3139,13	1,471	3186	0	3186	0
	95168	93093,5	2,180	95168	0	95168	0
	2139	2079,93	2,762	2139	0	2139	0
	776	583,767	24,772	776	0	776	0
	1035	1018,13	1,630	1035	0	1035	0
pet	87061	86760,1	0,346	87061	0	87061	0
	4015	4015	0	4015	0	4015	0
	6120	6091	0,474	6120	0	6120	0
	12400	12380,3	0,159	12400	0	12400	0
	10618	10560,9	0,538	10609,1	0,084	10608,6	0,089
	16537	16373,9	0,986	16528,1	0,054	16528,3	0,053
sento	7772	7606,03	2,135	7772	0	7772	0
	8722	8569,7	1,746	8721,2	0,009	8722	0
weing	141278	141263	0,011	141278	0	141278	0
	130883	130857	0,020	130883	0	130883	0
	95677	94496,2	1,234	95677	0	95677	0
	119337	118752	0,490	119337	0	119337	0
	98796	97525,3	1,286	98796	0	98796	0
	130623	130590	0,025	130623	0	130623	0
	1095445	1086484,2	0,818	1094579,6	0,079	1095432,7	0,001
	624319	581683	6,829	623727	0,095	624319	0
weish	4554	4530,03	0,53	4554	0	4554	0
	4536	4506,8	0,64	4536	0	4536	0
	4115	4009,4	2,57	4115	0	4115	0
	4561	4131,1	9,43	4561	0	4561	0
	4514	4159,7	7,85	4514	0	4514	0
	5557	5491,7	1,17	5557	0	5557	0
	5567	5428,4	2,49	5567	0	5567	0
	5605	5509,4	1,7	5605	0	5605	0
	5246	5104,5	2,7	5246	0	5246	0
	6339	6014,2	5,12	6339	0	6339	0
	5643	5234,3	7,24	5643	0	5643	0
	6339	5916	6,7	6339	0	6339	0
	6159	5769,5	6,3	6159	0	6159	0
	6954	6495,6	6,6	6954	0	6954	0
	7486	6684,6	10,7	7486	0	7486	0
	7289	6878,4	5,6	7289	0	7289	0
	8633	8314,7	3,7	8629,5	0,041	8633	0
	9580	9146,5	4,52	9559,63	0,213	9568,63	0,119
	7698	7223,2	6,2	7698	0	7698	0
	9450	8632,1	8,65	9448,63	0,014	9449,37	0,01
	9074	8114,4	10,6	9073,23	0,008	9073,33	0,01
	8947	8321,2	6,99	8926,73	0,227	8938,83	0,09
	8344	7603,8	8,87	8321,97	0,264	8318,93	0,3
	10220	9685,8	5,23	10152,9	0,657	10164,2	0,55
	9939	9077,9	8,66	9900,07	0,392	9910,73	0,28
	9584	8728,9	8,92	9539,4	0,465	9560,53	0,24
	9819	8873,7	9,63	9777,9	0,419	9802,03	0,17
	9492	8653,6	8,83	9423,87	0,718	9442,17	0,52
	9410	8466,7	10,02	9359,5	0,537	9369,5	0,43
	11191	10250,1	8,41	11106,3	0,757	11128,7	0,56

Table 5.2: Results of NSR, RSR and Time parameters obtained by GA, GA-SA and GA-SLS.

	GA			GA-SA			GA-SLS		
	NSR	RSR	Time	NSR	RSR	Time	NSR	RSR	Time
hp	1	1,7	1,8	**2**	**100**	6,1	**2**	**100**	7,1
pb	4	3,3	1,8	**6**	**100**	3,4	**6**	**100**	4,7
pet	4	32,8	1,4	**6**	**81,7**	11,3	**6**	80,5	12,2
sento	0	0,001	2,6	**2**	66,7	46,6	**2**	**100**	24,3
weing	6	39,6	1,7	**8**	76,3	10,3	**8**	**99,6**	10,6
weish	3	4,5	1,6	25	66,9	17,3	**26**	**76,9**	17,1
Average	18	13,6	1,82	49	81,9	15,8	**50**	**92,8**	**12,7**

5.3.3.1 Results for the SAC-94 standard instances

The average fitness (*Result*), the average gap (*GAP*), the best (*Best*) and the worst fitness (*Worst*), the number of success runs (*NSR*), the number of success instance (*NSI*) and the rate of success runs (*RSR*) have been recorded by analysing the recorded fitness. Also, the average CPU run-time (*Time*) has been calculated. All the results and statistics computed by the GA, GA-SA and GA-SLS are reported in Tables 5.1 and 5.2. From results, GA resolved to optimality one instance of 54 with average gap of 4,454 %, GA-SA 35 instances with a global gap of 0,093 % and GA-SLS 39 instances with a global gap of 0,0221 %. GA-SLS reached the optimum at least once in 50 instances followed by GA-SA in 49 instances then GA in 18 instances. The *RSR* show that GA-SLS totally solved instances of groups *hp, pb* and *sento* followed by GA-SA. GA-SLS obtained a total *RSR* better than GA-SA (92,83% and 81,91%, respectively). At the same time, GA-SA and GA-SLS widely surpass GA (13,65%). *RSR* shows that hybridisation of GA with SA has improved the success rate of 79,18% and its hybridisation with SLS of 68,49%. From Table 5.2, GA is the fastest with an global average CPU time of 1.818 sec.

5.3.3.2 Results for the ten large instances

From results on the GK shown in Table 5.3 GA-SA has the best value of *Result* and *GAP* for 1, 3, 5, 6 and 8 instances. GA-SLS has the best value of *Result* and

Table 5.3: Results of the approaches test on the GK data-set.

Dataset		GA		GA-SA		GA-SLS	
Instance	Optimal	Result	Gap	Result	Gap	Result	Gap
1	3766	3673,5	2,456	3704,3	1,638	3704,2	1,641
2	3958	3860,7	2,458	3894,8	1,596	3897,7	1,523
3	5656	5511,5	2,554	5538,8	2,072	5535,7	2,127
4	5767	5630,6	2,365	5655,2	1,938	5655,4	1,935
5	7560	7351,3	2,76	7395,1	2,181	7391,3	2,231
6	7677	7505,7	2,231	7528,4	1,935	7528,1	1,939
7	19220	18612,1	3,162	18691	2,752	18692,4	2,745
8	18806	18330,2	2,53	18393	2,196	18392,4	2,199
9	58091	56198,5	3,257	56371,1	2,96	56381,4	2,943
10	57295	55837,9	2,543	55959,3	2,331	55961,9	2,326
Average	18779,6	18251,2	2,632	18313,1	2,484	**18314,05**	**2,479**

GAP for instances 2, 4, 7, 9 and 10. Global, GA-SLS has the best performance for all instances with an total average *GAP* of 2.479 %. GA-SA has almost the same performance with average *GAP* of 2.484 %. Also, GA is not very far from GA-SA and GA-SLS with a total average *GAP* of 2.632 %.

5.3.3.3 Comparison with other GA approaches

We compared results of the proposed GA-SA and GA-SLS to other approaches. The results of the KHBA (Khuri et al., 1994), COTRO (Cotta & Troya, 1998), TEVO (Thiel & Voss, 1994), CHEBE (Chu & Beasley, 1998) and HGA (Djannaty & Doostdar, 2008) were obtained from (Djannaty & Doostdar, 2008). From Table 5.4 GA-SA and GA-SLS gave improved results compared to KHBA, COTRO and TEVO, for almost all instances. GA-SA and GA-SLS were able to find the optimal solutions to 6, and 3 of 7 problems respectively. Furthermore, GA-SA performs results quite similar to CHEBE and HGA.

Table 5.4: Comparison of GA-SA and GA-SLS with some GA-based approaches.

		KHBA	COTRO	TEVO	CHBE	HGA	GA-SA	GA-SLS
	Optimal	Sol A.	Sol A.	Sol A.	Sol A.	Sol A.	Sol A.	Sol A.
sento1	7772	7626	7767,9	7754,2	**7772**	**7772**	**7772**	**7772**
sento2	8722	8685	8716,3	8719,5	**8722**	**8722**	8721,2	**8722**
weing7	1095445	1093897	1095296	1095398	**1095445**	**1095445**	1094579	1095432
weing8	624319	613383	622048	622021	**624319**	**624319**	623727	**624319**
weish23	8344	8165,1	8245,8	8286,7	**8344**	**8344**	8321,97	**8344**
hp1	3418	3385,1	3394,3	3401,6	**3418**	**3418**	**3418**	**3418**
pb2	3186	3091	3131,2	3112,5	**3186**	**3186**	**3186**	**3186**

5.4 Memetic Search Algorithm (MSA) for the MKP01

MSA is a hybrid method composed of two algorithms GA and the Stochastic local-search-Simulated Annealing algorithm (SLSA). In MSA, the operators of the GA have been modified and adapted to the MKP01 and its efficiency has been improved by SLSA. In this section, MSA is described.

5.4.1 The Stochastic Local-Simulated Annealing algorithm (SLSA)

SLSA is a local-search algorithm that combines components from SLS and SA (SLSA has been presented in Section 4.2.2). In MSA, SLSA is used as intensification method to improve the fitness of each offspring generated by the GA. SLSA is applied to offspring following a local steps that consists in: firstly, create a neighbour solution by selecting and adding an item, secondly, repair the solution infeasibility. The SLSA process is divided into two steps as follows:

- *Step 1.* Select an item I to add to the solution with the probability $wp \in [0,1]$. The chosen item is selected according to one of the following criteria:
  - Chosen randomly an item, then if it increases the solution fitness $f(X)$ then it will be packed, otherwise it will be accepted if the following

expression is true $r_1 < \exp^{(-\Delta f/T)}$ where r is a random value, $\Delta f = f(X') - f(X)$ and T is a temperature value initially equal to T_0 relatively high.

- I is the best item i.e. the item having the highest value.

- *Step 2.* In a case of unfeasible solution, the worst item is discarded from the solution. This process is repeated until eliminated all conflicts. Then, the temperature value is updated. In our case, the decreasing rule is found empirically.

5.4.2 The GA population-based component of MSA

The MSA algorithm starts by launching the GA process. The GA steps and some modifications concerning selection, crossover, mutation, replacement operations are presented in following.

1. *Population initialisation.* MSA begins by the creation of the initial population P of individuals of a population size (PS). The RK method is used to create the individuals of P.

2. *Genetic operators.* The MSA selection operator is based on the fitness. The individuals in the population with higher values are selected. Two parents X_1,X_2 are selected to produce two new offspring. The crossover operator is based on the multiple point crossing method. Two new offspring X'_1,X'_2 are generated from this operator. Then, the crossing parents are stored in a list Q of size WT to prevent selecting the same individual more then once for a number of generation. The mutation is based on the multiple point flip method. Here, the mutated item is replaced by another one chosen randomly from the best item of the population. The number of mutation points is defined experimentally.

Algorithm 8 The MSA pseudo-code

```
Require: an instance of the MKP01. a vector of items that maximizes the knapsack revenue, empty queue $Q$ = ;
Ensure: the subset of items to be packed in the knapsack that maximize the profit.
 1: Generate randomly an initial population P according to the RK encoding
 2: while the maximum number of generations is not reached do
 3:     Select the two best individuals $X_1, X_2 \in P$ and $\notin Q$
 4:     Apply the crossover to obtain the new individuals $X'_1, X'_2$
 5:     Remove redundancy from $X'_1, X'_2$
 6:     Remove conflict from $X'_1, X'_2$
 7:     Apply the SLSA on $X'_1, X'_2$
 8:     Apply mutation on $X'_1, X'_2$ with $X_{best}$
 9:     if $(f(X'_1) > f(X_{worst}))$ then
10:         $P = P - \{X_{worst}\}$
11:         $P = P \cup \{X'_1\}$
12:     end if
13:     if $(f(X'_2) > f(X_{worst}))$ then
14:         $P = P - \{X_{worst}\}$
15:         $P = P \cup \{X'_2\}$
16:     end if
17:     $Q = Q \cup \{X_1, X_2\}$
18: end while
19: Return the best individual found.
```

3. a local-search improvement is performed on the new offspring X'_1, X'_2 using the SLSA (Algorithm of section 5.4.1). During this operation the offspring are repaired.

4. the offspring is compared to the worst individuals in the population in terms of fitness, they replace them in the population if they are better.

The optimisation process is repeated until the stopping criterion is verified. The criterion for stopping the optimisation process is a limited number of iteration NI determined empirically according to the size of the studied problem.

The steps of the MSA algorithm are described by the pseudo code in Algorithm 8.

5.4.3 Simulation results

We coded MSA in C++ and compile it on a PC having 2 GHz Intel Core 2 Duo processor and 2 GB RAM. To evaluate the efficiency and performance of our MSA algorithm, it was initially tested on 54 standard test problems (divided into six different sets) which are available at the OR-Library [2] maintained by Beasley. These problems are real-world problems widely used for the validation of the effectiveness of algorithms in the optimisation community. These problems consisting of $m = 2$ to 30 and $n = 6$ to 105. After several experiments, we set the parameters for the MSA as in Table 5.5.

Table 5.5: The values of MSA parameters.

Parameter	Description	Value
NI	Number of iteration	50000
PS	Population size	200
WT	Wait time	70
N	Number of iteration of SLSA	100
wp	Probability of SLSA	0,93
T_0	Initial temperature	30
CT	Coefficient T update	0.0105
Nrun	Number RUN	30

5.4.3.1 Results on SAC-94 data

Table 5.6 shows the results of MSA application on the 54 instances. Here columns n and m represent the number of items, and the number of constraints (the number of dimensions) respectively, column $Optimum$ is the value of Z optimal, column $Time$ is the average time of 30 runs. Column AVI is the average number of iteration of all the 30 runs. Column DTO is the deviation to the optimum and finally NSR is the number of successful runs.

From these results, we have identified various remarks. Firstly, MSA has succeeded to reach the optimal value once at least for all instances and 52 of the 54 in all 30 runs. The total average deviation of optimum is 0.0017%. Secondly,

[2] http://www.cs.nott.ac.uk/ jqd/the MKP01/

Table 5.6: Results of MSA applied on the SAC-94.

	n	m	*Optimum*	*Time*	*AVI*	*DTO*	*NSR*
HP	28	4	3418	5,020	24291	0	30/30
	35	4	3186	2,382	11237	0	30/30
PB	27	4	3090	1,133	5289	0	30/30
	34	4	3186	1,718	7717	0	30/30
	29	2	95168	0,335	1582	0	30/30
	20	10	2139	5,508	23963	0	30/30
	40	30	776	0,403	860	0	30/30
	37	30	1035	5,424	10746	0	30/30
PET	10	10	87061	0,013	1	0	30/30
	15	10	4015	0,010	2	0	30/30
	20	10	6120	0,013	13	0	30/30
	28	10	12400	0,905	3652	0	30/30
	39	5	10618	37,801	148	0,0922	9/30
	50	5	16537	13,059	47317	0,0026	29/30
SENTO	60	30	7772	0,773	1767	0	30/30
	60	30	8722	24,457	48509	0	30/30
WEING	28	2	141278	0,028	82	0	30/30
	28	2	130883	0,014	19	0	30/30
	28	2	95677	0,009	4	0	30/30
	28	2	119337	0,217	990	0	30/30
	28	2	98796	0,109	542	0	30/30
	28	2	130623	0,019	63	0	30/30
	105	2	1095445	58,112	121421	0	30/30
	105	2	624319	16,681	61761	0	30/30
WIESH	30	5	4554	0,010	10	0	30/30
	30	5	4536	0,057	216	0	30/30
	30	5	4115	0,063	248	0	30/30
	30	5	4561	0,013	28	0	30/30
	30	5	4514	0,013	35	0	30/30
	40	5	5557	0,647	2798	0	30/30
	40	5	5567	0,659	2863	0	30/30
	40	5	5605	0,893	3795	0	30/30
	40	5	5246	0,105	437	0	30/30
	50	5	6339	0,619	2491	0	30/30
	50	5	5643	0,774	3251	0	30/30
	50	5	6339	0,706	2810	0	30/30
	50	5	6159	0,782	3222	0	30/30
	60	5	6954	3,623	13939	0	30/30
	60	5	7486	0,613	2326	0	30/30
	60	5	7289	2,767	10593	0	30/30
	60	5	8633	9,375	31597	0	30/30
	70	5	9580	15,670	53919	0	30/30
	70	5	7698	4,035	15589	0	30/30
	70	5	9450	3,713	12805	0	30/30
	70	5	9074	3,445	11963	0	30/30
	80	5	8947	7,761	26499	0	30/30
	80	5	8344	32,683	112813	0	30/30
	80	5	10220	60,773	192505	0	30/30
	80	5	9939	29,397	96748	0	30/30
	90	5	9584	14,307	47672	0	30/30
	90	5	9819	20,250	67586	0	30/30
	90	5	9492	23,620	78641	0	30/30
	90	5	9410	18,233	61025	0	30/30
	90	5	11191	232,352	683614	0	30/30

Table 5.7: WT influence with PS = 50 individuals.

	5	6	8	10	12	16	25	**33**
AFV	16464,4	16453,1	16465,3	16510,6	16484,4	16515,1	16527,3	**16530,4**
BFV	16524	16518	16518	16537	16537	16537	16537	**16537**
WFV	16424	16395	16417	16457	16439	16439	16510	**16515**
DEV	0,439	0,507	0,433	0,159	0,318	0,132	0,058	**0,039**
AVI	44847	46151	46774	45551	45206	45186	44972	**45018**
NOP	0	0	0	2	1	4	5	**6**

in 27 of 54 instances, the average execution time of MSA is less than one second, however, some instances required more time than the rest as WEISH 23-29, PET 5, WEING 7 and 8 and, in particular, the WEISH30, but in general, the time is relatively little with global average of 11,7 seconds. 7,4 seconds if the WEISH30 is ignored. Thirdly, the number of required average iteration varies from 1 to 600 thousand iterations (for PET1 and WEISH30 respectively) or an average of 35895 iterations and 23194 iterations without the WEISH30.

5.4.3.2 Impact of the Waiting Time parameter (WT)

The Waiting Time WT is the mechanism that synchronises the participation of individuals in the MSA algorithm operations. It is a waiting list of size relative to the size of the population (PS). In order to assess its influence, MSA is tested for 8 values of WT calculated the ratio PS * k such that k = 1/10, 1/8, 1/6, 1/5, 1/4, 1/3, 1/2, 2/3. The influence of WT on the behaviour of MSA is evaluated for three sizes of population PS = 50, 100, 200. Finally, the following factors have been identified: the average fitness "AVF", the best fitness "BFV", the worst fitness "WFV", the deviation to optimum "DEV", the average number of iteration "AVI" necessary to obtain optimum and finally the number of times that the optimal value is obtained "NOP" (Number of Optimal case). TW is tested with several PS values $\{50, 100, 200\}$ given in Tables 5.7, 5.8 and 5.9 respectively. The number of optimal solutions is given in Table 5.10.

Table 5.10 summarises Table 5.7, 5.8, 5.9 in terms of number of successful runs.

Table 5.8: WT influence with PS = 100 individuals.

	10	12	16	20	25	33	**50**	66
AFV	16483.2	16486.8	16500.7	16521.7	16518.5	16527.1	**16531**	16527.8
BFV	16537	16524	16524	16537	16537	16537	**16537**	16537
WFV	16433	16439	16447	16477	16417	16464	**16490**	16508
DEV	0.325	0.303	0.219	0.092	0.111	0.059	**0.036**	0.055
AVI	45827	45092	45251	43487	43470	44170	**44916**	45006
NOP/10	1	0	0	3	4	7	**8**	5

Table 5.9: WT influence with PS = 200 individuals.

	20	25	33	40	**50**	66	100	133
AFV	16511,4	16516,5	16518	16527,7	**16534,4**	16529,7	16522,6	16519.8
BFV	16537	16537	16537	16537	**16537**	16537	16537	16537
WFV	16439	16463	16470	16470	**16524**	16507	16496	16508
DEV	0,154	0,123	0,114	0,056	**0,015**	0,044	0,087	0.104
AVI	44166	43818	42864	42320	**41788**	41966	42966	43487
NOP/10	5	4	1	7	**8**	6	2	2

Table 5.10: The number of optimal solutions for different values of WT.

PS	1/10	1/8	1/6	1/5	1/4	**1/3**	1/2	2/3	total
50	0	0	0	2	1	4	5	6	18
100	1	0	0	3	4	7	8	5	28
200	5	4	1	7	8	6	2	2	**35**
SUM	6	4	1	12	13	**17**	15	13	

There we can see that in 15 of the 24 possible cases MSA has failed to achieve the optimum in more than 5/10 runs. MSA has reached the optimum in 5/10 runs in only 3 of the 24 possible cases and more than 5/10 runs in only 6/24 possible cases. In addition, the most appropriate population size is PS = 200 to which corresponds a WT size between 66 and 100 (we set WT = 70). Finally, we can conclude that WT is a determining factor in the operation of crossover and therefore greatly influences the effectiveness of the MSA algorithm.

5.4.3.3 MSA compared to other approaches

1. *MSA vs. GA, GA-SA and GA-SLS.*

   In this experiment the MSA approach was compared to three other approaches: the Genetic Algorithm (GA), its hybridisation with the Simulated

Table 5.11: The average deviation obtained by MSA for PS equals to: 50, 100 and 200 individuals.

PS	1/10	1/8	1/6	1/5	1/4	1/3	**1/2**	2/3
50	0,439	0,507	0,433	0,159	0,318	0,132	0,058	**0,039**
100	0.325	0.303	0.219	0.092	0.111	0.059	**0.036**	0.055
200	0,154	0,123	0,114	0,056	**0,015**	0,044	0,087	0.104
AVERAGE	0,306	0,311	0,255	0,102	0,148	0,078	**0,060**	0,066

Annealing (GA-SA) and by the Stochastic local-search (GA-SLS) (Rezoug et al., 2015). In all tests, same parameters were used. We applied the GA, GA-SA, GA-SLS and MSA algorithms on the WEISH27 instance (optimum = 9819). The value of fitness was noted every second for 15 seconds. With the average of fitness obtained in 10 runs, the curves 5.2 and histogram 5.3 were drawn.

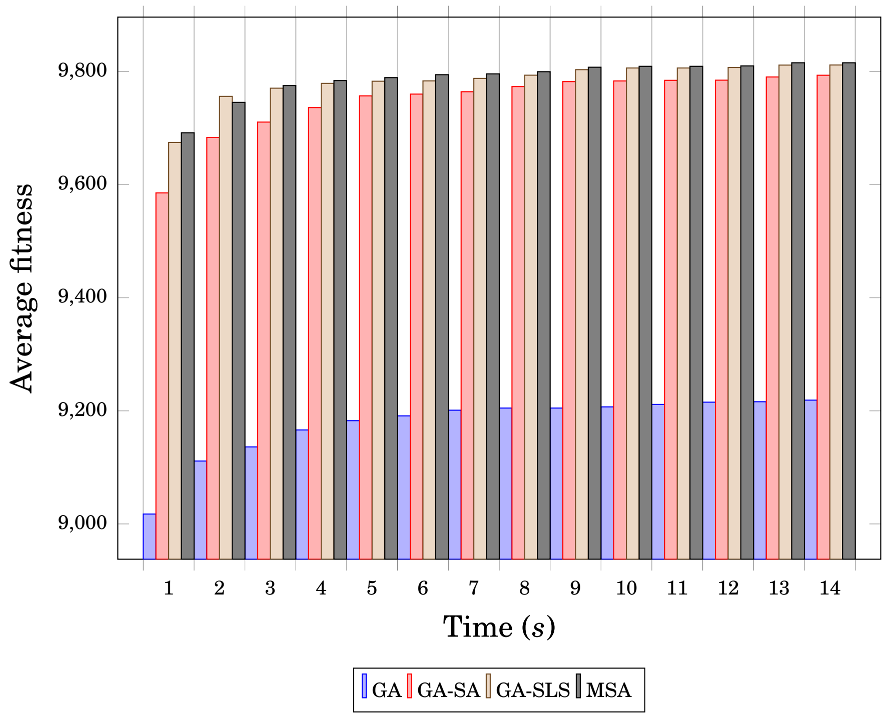

Figure 5.2: Comparing MSA to GA, GA-SA, GA-SLS through the WEISH27

The curves of the Fig. 5.2 and 5.3 represent the evolution of the four algorithms for 15 seconds. Algorithms start using population generated according

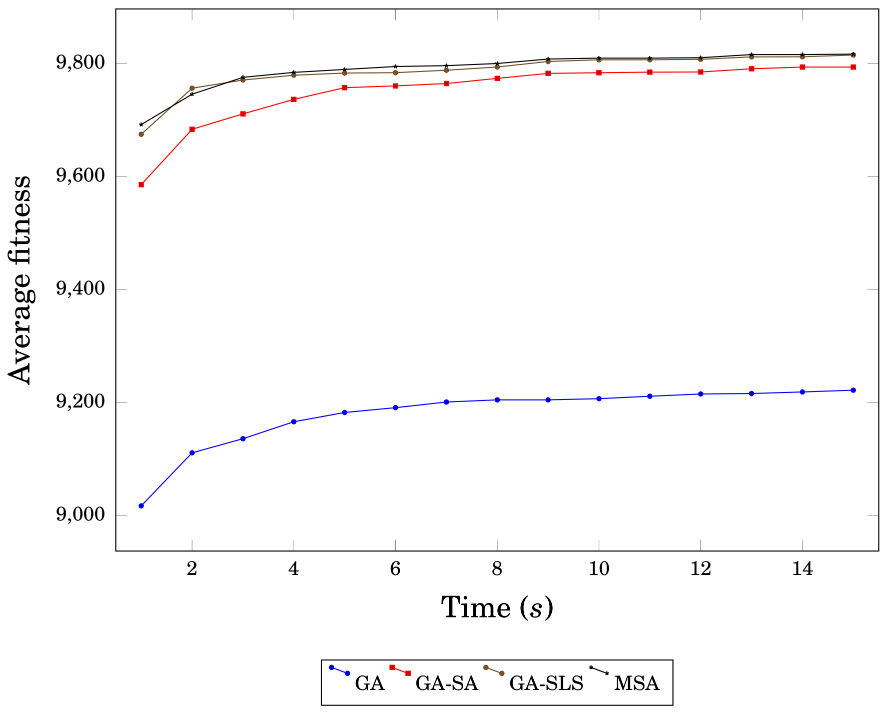

Figure 5.3: Comparing MSA to GA, GA-SA, GA-SLS through the WEISH27

to RK. The algorithms MSA and GA-SLS curves are too close together except at the end where only MSA reaches the optimum.

These two algorithms exceeded slightly that of the GA-SA algorithm and largely that of the GA algorithm. We can deduce that MSA is faster and more efficient than the other three algorithms. Indeed, the same experience with WEISH8 (optimal = 624319). But this time, the value of fitness was relieved each 0.5 seconds for 25 seconds, confirms the results of the first experiment. The Fig. 5.4 shows the results.

2. *MSA vs. CRGA and SRGA.*

   Table 5.12 shows the comparison of MSA with CRGA and SRGA algorithms (H. Yang et al., 2013) in terms of effectiveness. Where CRGA and SRGA are two algorithms based on GA. The comparison was done according to the calculated means fitness. The values of results presented in Table 5.12 are

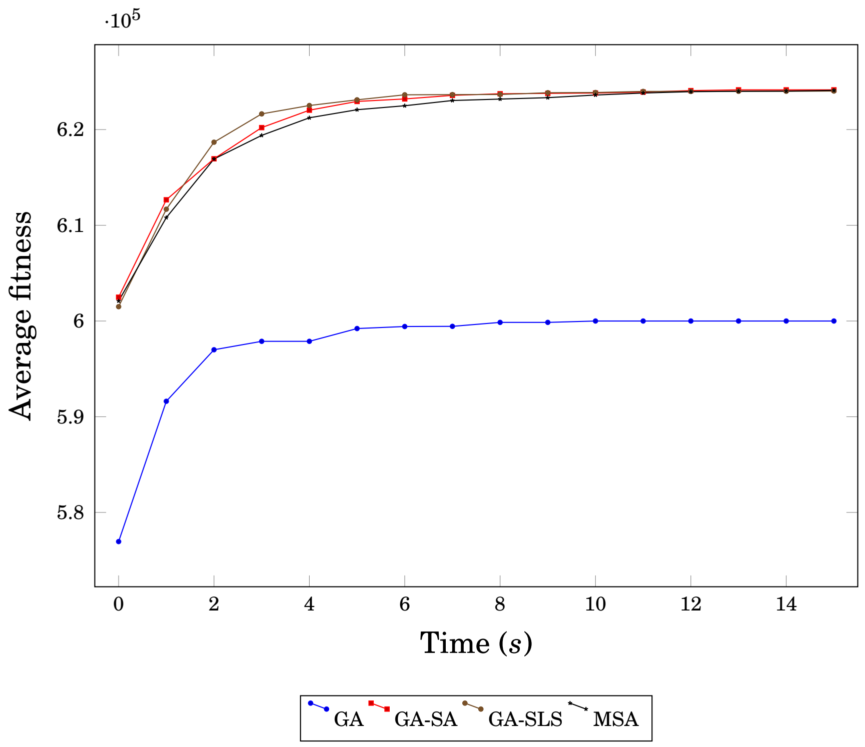

Figure 5.4: Comparing MSA to GA, GA-SA, GA-SLS through the WEING8.

the same published by the authors.

We can observe that MSA was able to reach the optimum in almost all instances, whereas CRGA and SRGA were not able to reach the optimum in any instance. MSA is more effective than CRGA and SRGA.

3. *MSA vs. DPHEU.* Table 5.13 shows the comparison of MSA with DPHEU algorithm (Veni & Balachandar, 2010) in terms of effectiveness. DPHEU is. The comparison was done according to the calculated average p o d APOD and the number NOPT. The values of results presented in Table 5.13 are the same published by the authors.

   Both MSA and DPHEU obtained similar results in three of the six sets (HP, SENTO and WEING). MSA is more effective than DPHEU in two sets PB and WEISH. DPHEU is more effective than MSA in the PET set. We can say that MSA and DPHEU are quite similar in terms of effectiveness.

Table 5.12: Mean fitness obtained by MSA compared to CRGA and SRGA.

instance	optimum	CRGA Mean	SRGA Mean	MSA Mean
PET1	3800	3782	**3800**	**3800**
PET2	8706	8562	8662	**8706**
PET3	4015	3878	3941	**4015**
PET4	6120	5476	5630	**6120**
PET5	12400	11203	12240	**12400**
PET6	10618	10107	9953	**10608,2**
PET7	16537	15184	14915	**16536,5**
HP1	3418	3259	3214	**3418**
HP2	3186	2921	2864	**3186**
Weing1	141278	130885	131409	**141278**
Weing2	130883	113289	116883	**130883**
Weing4	119337	107535	106950	**119337**
Weing5	98796	79038	75109	**98796**
Weing6	130623	116773	115671	**130623**
Weing7	1095445	975269	783196	**1095445**
PB1	3090	2953	2936	**3090**
PB2	3186	2965	2907	**3186**
PB4	95168	83483	81412	**95168**
PB5	2139	1984	2016	**2139**
Weish1	4554	3774	3777	**4554**

Table 5.13: Average fitness and number of optimal solutions obtained by MSA compared to DPHEU.

data	number of instances	DPHEU		MSA	
		A.P.O.D	N.O.P.T	A.P.O.D	N.O.P.T
HP	2	0.0	2	0.0	2
PB	6	0.04	5	**0.0**	**6**
PET	6	**0.0**	**6**	0,0158	4
SENTO	2	0.0	2	0.0	2
WEING	8	0.0	8	0.0	8
WEISH	30	0.03	28	**0.0**	**30**

4. *Further comparison.* The MSA effectiveness is more demonstrated through the comparison to the works by (Cotta & Troya, 1998; Chu & Beasley, 1998; Djannaty & Doostdar, 2008; Khuri et al., 1994; Thiel & Voss, 1994) shown in Table 5.14. MSA, HGA and CHBE resolved to optimality all instances used in the tests.

Table 5.14: Fitness obtained by MSA compared to other approaches based on GA.

data	optimum	KHBA Sol A.	COTRO Sol A.	TEVO Sol A.	CHBE Sol A.	HGA Sol A.	MSA Sol A.
sento1	7772	7626	7767.9	7754.2	7772	7772	**7772**
sento2	8722	8685	8716.3	8719.5	8722	8722	**8722**
weing7	1095445	1093897	1095296.1	1095398.1	1095445	1095445	**1095445**
weing8	624319	613383	622048.1	622021.3	624319	624319	**624319**
weish23	8344	8165.1	8245.8	8286.7	8344	8344	**8344**
hp1	3418	3385.1	3394.3	3401.6	3418	3418	**3418**
pb2	3186	3091	3131.2	3112.5	3186	3186	**3186**

5.5 Adaptive Harmony Search for the MKP01

It has proven advantageous to combine a population based method with a local-search to ensure a balance between the global exploration and the local exploitation of the search space. Motivated by this idea, we propose a hybrid Self-Adaptive Harmony Search (SAHS) combined with SLS to solve the MKP01. SAHS is used to ensure exploration while SLS performs exploitation. First, we improved the SAHS by adding a tuning strategy for the pitch adjusting rate (PAR) and the Bandwidth (bw). then, we apply the local-search SLS on every generated solution with a specified Probability (ESP) strategy. the proposed SAHS-SLS is evaluated on the well-known the MKP01 benchmarks proposed by (Chu & Beasley, 1998).

5.5.1 Self-adaptive Global best Harmony Search algorithm (SGHS)

Self-adaptive Global best Harmony Search algorithm (SGHS) (Pan et al., 2010) proposed a self-adapted tuning of the parameters *HMCR*, *PAR* and bw by a learning mechanism, a modified *PAR* and dynamic generation of bw. SGHS proposed also, in the memory consideration phase, an improved method to avoid getting trapped in local optima. SGHS can be summarised by the pseudo code in Algorithm 9.

Algorithm 9 the SGHS method.

```
Require: $HMS$,$LP$,$NI_{max}$,$n$,$bw_{max}$,$bw_{min}$,$HMCRm$ and $PAR$
Ensure: the better feasible solution $X_{best}$
 1: initialise and evaluate HM. Set generation counter $lp = 1, NI = 0$.
 2: for $i = 1$ to $NI$ do
 3:     Generate $HMCR$ and $PAR$ according to $HMCRm$ and $PARm$. Yield $bw$
        according to $bw_{max}$ and $bw_{min}$.
 4:     for $j = 1$ to $n$ do
 5:         if $r \leq HMCR$ then
 6:             $x_{newj} = x_{aj} \pm r1 * bw. where\, a \in \{1, \cdots, HMS\}$;
 7:             if $r2 \leq PAR$ then
 8:                 $x_{newj} = x_{bestj}$;
 9:             end if
10:         else
11:             $x_{newj} = x_L + (x_U - x_L) * r3$;
12:         end if
13:     end for
14:     if $f(x_{new}) < f(x_{worst})$ then
15:         update the $HM$ as $x_{worst} = x_{new}$ ; Record the values of $HMCR$ and $PAR$.
16:     end if
17:     if $lp = LP$ then
18:         recalculate $HMCRm$, $PARm$ according to the recorded values of $HMCR$,
            $PAR$ and reset $lp = 1$;
19:     else
20:         $lp = lp + 1$;
21:     end if
22: end for
23: Note: $where\, r, r1, r2 \in [0, 1]$
```

5.5.2 Hybrid Self-Adaptive Harmony Search (SAHS-SLS)

SAHS-SLS initialises the HM population following the RK method before starting the optimisation. After that, the optimisation step consists in improvising a new harmony according to SAHS, then with probability P, apply a local-search SLS.

5.5.2.1 *bw* tuning strategy

In the improvisation step, bw is generated in two phases. First, bw is computed according to the number of iterations (NI). bw starts with a high value and decreases continuously until half NI which allows to explore the search space. In the second phase, for each improvised harmony X_j corresponds a bw_j value set at random in $[bw_{min}, bw_{max}]$. the bw is generated as the following strategy:

$$bw_j = \begin{cases} \dfrac{bw_{max} - bw_{min}}{NI} & i \leq NI/2 \\ bw_{rand} \quad \in [bw_{max}, bw_{min}] & i > NI/2 \end{cases}$$

5.5.2.2 Modified Pitch Adjusting Rate (*PAR*).

In SAHS-SLS, the *PAR* is modified as follows: the selected item from X_{best} of HM is slightly adjusted with a bw generated according to the strategy seen previously. the adjustment is applied with a probability $pbw2$. A small value of *PAR* guaranties the compromise between exploitation and exploration. Contrary, a high value increases the exploration but decreases the exploitation and the obtained solutions will be of bad quality. The proposed *PAR* is given in Algorithm 10 by lines 8-13.

5.5.2.3 *WP* adaptation

the *WP* parameter is dynamically generated in SAHS-SLS. At each iteration, a value of *WP* is generated according to a normal distribution with a mean value *WPm* and deviation (0.01). the values of *WP* allowing SAHS-SLS to obtain a better harmony than the worst harmony among HM are saved. After a number of iterations *LP*, *WPm* is adjusted to the average of the saved best *WP* values.

5.5.3 Simulation results

the proposed algorithms were coded in C++. the experiments were run on an Intel core 2 duo CPU 2 GHz and 2 GB of RAM and windows 7 32-bit. Large the MKP01 data set available in the OR-library (Chu & Beasley, 1998) have been used. the problem set consists of 270 instances. there are 9 classes each one composed of 3

Algorithm 10 the SAHS-SLS algorithm.

```
Require: Set HMS, NI, n, HMCRm, PAR, WPm, LP, P,MI, pbw1, pbw2, $bw_{min}$
    and $bw_{max}$.
Ensure: the better feasible solution $X_{best}$
 1: initialise HM according to RK and set $lp = 1, NI = 0$.
 2: for $i = 1$ to $NI$ do
 3:   Generate HMCR, PAR and WP according to HMCRm, PARm and WPm.
      Yield $bw$.
 4:   improvise a new harmony $X_{new}$ as follows :
 5:   for $j = 1$ to $n$ do
 6:     if $r \le HMCR$ then
 7:       $x_{newj} = x_{ak} \pm pbw1 * bw. where\, a \in \{1, \cdots, HMS\}$;
 8:       if $r1 \le PAR$ then
 9:         $x_{newj} = x_{bestj} \pm pbw2 * bw$.
10:       end if
11:     else
12:       $x_{newj} = x_{rand}$;
13:     end if
14:   end for
15:   if $r2 \le P$ then
16:     apply SLS to the new harmony $X_{new}$
17:   end if
18:   if $f(X_{new}) > f(X_{worst})$ then
19:     Update HM as $X_{worst} = X_{new}$; Record the values of HMCR, PAR and wp
20:   end if
21:   if $lp = LP$ then
22:     Recalculate HMCRm, PARm and WPm according to the recorded values
        of HMCR, PAR and WP respectively; lp =1
23:   else
24:     lp = lp +1
25:   end if
26: end for
```

groups, 10 instances in every group. the number of constraints m and items n are 5, 10, 30 and 100, 250, 500 respectively. the optimal solution for some instances are yet unknown. We compared the obtained results to those collected in the web page [3].

[3] http://www.cs.nott.ac.uk/~jqd/mkp/results.html

Table 5.15: Impact of HMS in SAHS-SLS

		10	30	70	100	200	500
(5,10,30)x100_0.25	Deviation	1,092	**0,718**	0,763	0,789	0,708	1,054
.	Time	34,063	36,676	50,091	46,386	54,681	163,209
(5,10,30)x250_0.25	Deviation	2,016	**1,831**	1,865	1,891	2,014	2,541
.	Time	68,337	71,698	81,109	83,943	98,091	191,815
(5,10,30)x500_0.25	Deviation	2,538	**1,948**	1,964	1,982	2,399	3,423
.	Time	166,127	159,143	174,019	177,917	194,035	328,02
Average	**Deviation**	1,882	**1,499**	1,530	1,554	1,707	2,339
.	**Time**	89,509	89,172	101,739	102,748	115,602	227,681

Table 5.16: the appropriate probability of SLS application.

	0	0.1	0.2	0.4	0.8	0.9	1
Deviation(%)	1.22	0.6	0.58	0.68	**0.52**	0.59	0.64

5.5.3.1 Parameters tuning

This experiment aims to determine the impact of HMS and P on SAHS-SLS. the parameters of the proposed algorithms are empirically fixed. In each experiment, only the tested parameter is changed and one the MKP01 instance is used. the average fitness and CPU time is taken over 20 runs. HMS varies between 10 and 100, and $P \in [0,1]$.

- **HMS**

  We used six different values of HMS to evaluation its impact on effectiveness and speed of SAHS-SLS. We used the ten first instances from each of the nine the MKP01 benchmark classes (those with $\alpha = 0.25$). the results showed in Table 5.15 represent the average value of CPU time and the fitness obtained for instances having the same number of items $m = 100$, $m = 250$ and $m = 500$.

- ***WP***

  This experiment was carried out only with the *OR10x250_0.75_1* first instance to determine, in an empirical way, the value of suitable probability for applying SLS following the ESP strategy. For that, the value of probability

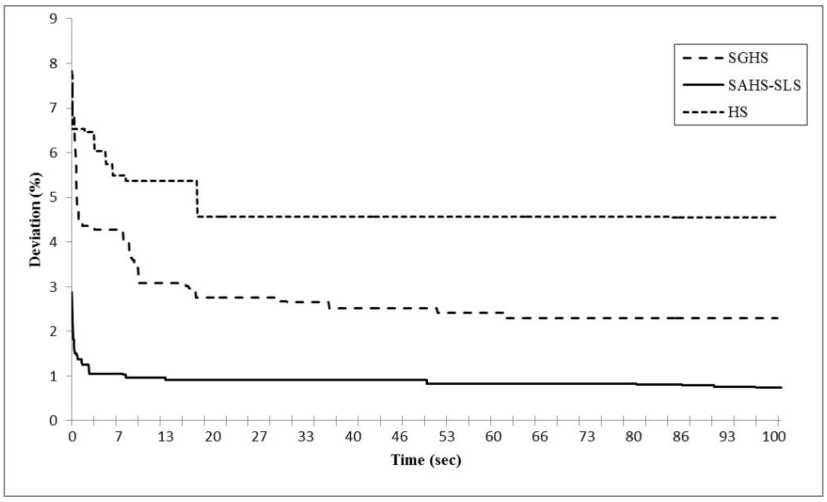

Figure 5.5: the convergence speed of SAHS-SLS, SGHS and HS.

was varied between seven different values. the results are reported in Table 5.16:

5.5.3.2 SAHS-SLS speed investigation

By using the same instance $OR10x250\_0.75\_1$ we carried out experimental tests in order to compare the speed of convergence of the three algorithms. It consists in running SAHS-SLS, SGHS and HS during 100 seconds. the value of fitness was recovered each 0.05 second. the obtained results were translated into the curves depicted in Fig. 5.5.

5.5.3.3 Results on large data

Constant values were assigned to the parameters to which the values are not dynamically generated. Table 5.17 summarises the values of those parameters for the three algorithms used to carry out the experiments. Table 5.18 reports the results of SAHS-SLS, SGHS and HS.

Table 5.17: the values of the algorithms parameters.

Parameter	Value
HMS	30
$[BW_{min}, BW_{max}]$	[1, 10]
pbw1 and pbw2	0.0001
P	0.8
HMCR	[0.9, 1]
PAR and wp	[0, 1]
HMCRm	0.99
PAR	0.8
WPm	0.7
NI	{30000, 50000, 75000, 100000}
n	{700, 1000, 1500}
maxiter	200
LP	200

5.6 Conclusion

This chapter aimed to present three approaches based on hybridisation between a population-based and a single solution-based methods. The studies revealed several conclusions on the performance of each approach.

The chapter began by describing two hybrid methods, GA-SA and GA-SLS, that consists of a GA combined with two local-search algorithms SA and SLS. In the current study, comparing GA-SA with GA-SLS showed that GA combined with SLS outperforms its combination with SA and gives a high-quality solutions.

After that, the chapter introduced the Memetic Search Algorithm (MSA) for dealing with MKP. It is a GA augmented by the local-search algorithm SLSA presented in Chapter 4. MSA was also able to achieve high-quality solutions and outperformed several other methods in literature.

Finally, the chapter presented a new approach resulting from the Self-Adaptive Harmony Search (SAHS) combined with SLS. The SAHS introduced a new strategy to measure bw and a new structure for the PAR part, and it has been combined with SLS. These changes allowed SAHS-SLS to achieve encouraging performances on complex and large data and surpass other methods.

Table 5.18: Comparison of SAHS-SLS, SGHS and HS.

			SAHS-SLS		SGHS		HS	
			Deviation	Time	Deviation	Time	Deviation	Time
5	100	0.25	0,6	18,89	0,77	16,33	5,35	9,54
		0.50	0,57	24,79	0,78	22,32	5,15	12,35
		0.75	0,32	29,06	0,47	27,56	3,09	16,55
Average			0,49	24,24	0,68	22,07	4,53	12,82
	250	0.25	2,14	31,54	2,62	26,56	8,41	22,39
		0.50	1,48	68,07	2,27	57,16	6,21	36,31
		0.75	1,215	99,34	1,77	80,45	3,78	62,32
Average			1,61	66,31	2,22	54,72	6,13	40,34
	500	0.25	1,85	146,3	3,75	109,06	9,57	66,74
		0.50	1,31	248,36	3,59	177,27	7,24	116,41
		0.75	1,33	339,97	2,68	269,62	4,67	187,63
Average			1,5	244,87	3,34	185,32	7,16	123,59
10	100	0.25	0,9	29,01	1,26	27,84	5,31	16,46
		0.50	0,64	46,6	0,97	33,61	4,89	20,58
		0.75	0,36	48,8	0,64	43,51	3,22	27,32
Average			0,63	41,47	0,96	34,99	4,48	21,46
	250	0.25	1,49	93,28	2,33	74,11	7,17	44,74
		0.50	0,91	160,19	2,06	116,26	6,01	76,59
		0.75	0,67	261,48	1,59	172,83	3,84	116,62
Average			1,03	171,65	1,99	121,07	5,67	79,32
10	500	0.25	2,11	157,14	4,16	118,5	9,95	78,03
		0.50	1,61	267,03	4,23	207,42	7,95	143,54
		0.75	1,45	384,42	4,11	297,31	5,04	229,34
Average			1,72	269,53	4,16	207,74	7,65	150,3
30	100	0.25	1,23	76,98	0,93	68,49	2,79	49,69
		0.50	0,68	64,93	0,96	52,54	3,9	36,46
		0.75	0,33	87,05	0,52	67,91	3,21	46,85
Average			0,75	76,32	0,8	62,99	3,3	44,33
30	250	0.25	1,93	155,91	2,47	91,9	7,59	61,82
		0.50	0,84	155,75	2,49	102,97	6,99	71,56
		0.75	0,66	231,71	1,77	150,69	4,28	108,27
Average			1,14	181,12	2,24	115,18	6,29	80,55
30	500	0.25	2,24	173,32	5,62	127,13	9,71	88,99
		0.50	1,19	264,16	4,82	206,27	8,39	137,07
		0.75	0,87	343,99	6,31	285,05	6,43	243,26
Average			1,43	260,49	5,58	206,15	8,18	156,44
Average all			1,14	148,44	2,44	108,64	6,01	74,01

CHAPTER 6

KNOWLEDGE-GUIDED GA FOR THE MKP01

6.1 Introduction

Genetic Algorithm (GA) has emerged as an efficient method for solving wide range of Combinatorial Optimisation (CO) problems. Its application has long been a question of great interest in a wide range of research fields.

Genetic Algorithm (GA) was first introduced more than four decades ago and it is still a widely used in several research applications. GA is mainly used as a stochastic method for solving CO problems, especially for NP-hard problems. In may real-world problems, exact methods fail to find a satisfactorily solution in a reasonable processing time. GA has been applied successfully in many real applications as well as many traditional combinatorial problems (Gen, 2006). Originally, GA is inspired by the biological evolution of living species. Starting with a randomly generated initial population of a set of individuals, GA aims to improve the quality of the successive generations by a set of selection and modification operators, e.g. crossover and mutation. It is known that GA is relatively simple to implement compared to several other methods, but, is it really able to provide the best solutions? In reality, GA is a stochastic process, so there is no guarantee of optimality, only a large number of generations and individuals can increase the confidence in the obtained solution (Pappa & Freitas, 2009; Snášel et al., 2010).

This chapter presents a GA variant - named Guided GA (GGA) - for the Multi-dimensional Knapsack Problem (MKP01). The proposed algorithm is inspired by two main concepts. The first is the concept of Proximate Optimality; in most cases, the best solutions have a similar structure, in other words; part of the solution may appear in all the best individuals. The second motivation is based on the Core Concept for the Multidimensional Knapsack Problem CCMKP01 (Puchinger et al., 2010; Senju & Toyoda, 1968); CCMKP01 provides a mathematical model for ordering the items in the MKP01 based on a compromise between the cost and the benefit of each object. GGA uses the output of the CCMKP01 model as an additional guide for the GA's evolutionary process. The CCMKP01 output is used at two stages of the evolutionary process; in the initialisation and evaluation (fitness function) stages.

The chapter is structured as follows: Section 6.3 gives an overview of the literature review related to the GGA. The proposed algorithm GGA is introduced in Section 6.4. The experimental setup and the parameters tuning are given in Section 6.5. Section 6.6 presents the conducted experiments and the obtained results. The conclusions and final remarks are drawn in Section 2.5.

6.2 The Core Concept for MKP01

The core concept for solving the combinatorial and linear programming problems was specifically applied for the knapsack problems by (Balas & Zemel, 1980) and successfully extended to the MKP01 by (Puchinger et al., 2010). It consists in the part of variables (items) for which it is hard to approximately know in advance what their value will be in an optimal solution. The CCMKP01 calculates a score for each item in the MKP01, the score reflects the expected added value of the item to the final solution; a high score indicates that the item is likely to appear in the optimal solution, low score indicates that the item is unlikely to appear in the optimal or near-optimal solutions, while "average" score indicate that there is uncertainty about the item's added value. Therefore, after being sorted decreasingly according to CCMKP01 score, the variables are divided into three sets. The variables with high efficiency measures are fixed to 1 whereas those with low efficiency measures are fixed to 0 and those with close efficiency represent the core. Consequently, the core concept allows reducing the original problem into only the core problem.

$$e_j^{simple} = \frac{p_j}{\sum_{i=1}^{m} w_{ij}} \tag{6.1}$$

$$e_j^{scaled} = \frac{p_j}{\sum_{i=1}^{m} \frac{w_{ij}}{c_i}} \tag{6.2}$$

$$e_j^{st} = \frac{p_j}{\sum_{j=1}^{m} w_{ij}(\sum_{l=1}^{n} w_{il} - c_i)} \tag{6.3}$$

$$e_j^{general} = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}} \tag{6.4}$$

$$r_i = \frac{\sum_{j=1}^{n} w_{ij} - c_i}{\sum_{j=1}^{n} w_{ij}} \tag{6.5}$$

As mentioned earlier, the core concept is based on the efficiency measurement function (score). The aim is to assigns an efficiency value to each variable, according to its significance in producing the optimal solution, in such a way to promote those having the high benefits and low costs. Several efficiency definitions have been used as approximations of the efficiency function, for example, simple efficiency (e_j^{simple}) (Dobson, 1982), scaled efficiency (e_j^{scaled}), Senju & Toyoda (e_j^{st}) Senju and Toyoda (1968) and general efficiency ($e_j^{general}$) (Freville & Plateau, 1994; Kellerer et al., 2004b) as shown in Eq. 6.1, 6.2, 6.3 and 6.4-6.5 respectively.

Balas and Zemel (1980) provided a method based on the core concept to solve the knapsack problem. In their solution the authors supposed that the weights of the items are uniformly distributed. Thus, they claimed that if the weights in the core are uniformly distributed, then there is a high probability of finding an optimal solution in the core. However, Pisinger (1999) demonstrated experimentally that the weights in the core are not uniformly distributed in some complex test. Also, it is experimentally shown that the heuristic proposed by (Balas & Zemel, 1980) does not find as good solutions as expected.

The core concept method has many inconveniences related to the method itself or to the data used. Firstly, the solution quality depends on the core size which is different from an instance to another. Secondly, the more is the correlation between the efficiency of the variables the hard is to find the core. Thirdly, it is not easy to generalise the core concept method to multi-objectives combinatorial optimisation problems, when the variables are continuous or for the problems with multiple variables. Finally, if the adopted method to sort the variables is not efficient, then the obtained results using the core concept method will be of low quality.

6.3 Related Works

There are several methods related to the guided GA concept in the literature, that has been applied to a wide range of applications. For solving the Course Timetabling Problem, the approaches by (Jat & Yang, 2009; S. Yang & Jat, 2011) use a memory denoted MEM to record useful information to guide the GA process and improve its performance. MEM is a list of limited size, in which a list of room and time slot pairs is recorded. This information is integrated into the crossover operator of the proposed guided GA. Other researchers used an external structure to guide GA such as (Acan & Tekol, 2003; Louis & Li, 1997). Another approach for guiding the GA is through the use of approximate probabilistic models. In (S.-H. Chen et al., 2012; Q. Zhang, 2009) The GA is augmented with an approximate probabilistic model to guide the crossover and mutation operators. The probabilistic model is used to estimate the quality of candidate solutions generated by the traditional crossover and mutation operators. It also evaluates the quality of candidate solutions. This estimation enables the crossover and mutation operators to generate more promising solutions.

Specific characteristics of the addressed problem are used to guide the GA search process. The Process Discovery through a Genetic algorithm (ProDiGen) (Vázquez-Barreiros et al., 2014) is a GA that adopts three characteristics of the Process Discovery. A methods calculate the precision, simplicity and completeness values of the treated model (i.e. log files of the information system process). These values are integrated into the expression of the GA fitness function to guide the optimisation process. A slowdown-guided GA for the job scheduling problem is proposed by (Gabaldon et al., 2014). The proposed model is based on the estimation of the execution slowdown of the tasks which is used to guide the GA search process, the slowdown estimation is embedded the fitness function.

A subset of the genetic operators is guided. The proximate optimality principle assumes that good solutions have a similar structure. Based on this principle, the guided mutation proposed by (Q. Zhang et al., 2005) uses a probability model inspired by estimation of distribution algorithms EDA mutation operator. The generated offspring by this operator is constructed based on the best parent so far and a dynamic probability model and a probability β. This allows conducting the searching process in promising areas. A guided crossover operator has been proposed by (Rasheed, 1999). The crossover operator works by using guidance from all members of the GA population to select a direction for exploration. The first parent is selected by the selection operator. To select the second parent, a metric named $Mutual\_fitness$ is calculated for all the other chromosomes. The chromosome which has the maximum value is selected. One offspring is generated by crossing the parents in a point chosen randomly such that the offspring resulting is the best.

6.4 The Guided Genetic Algorithm GGA

The algorithm in this chapter is motivated by the observation that in many optimisation real-world problem, we may have some prior information about the components/patterns that are likely to appear in the good solutions. For example, in MKP01, it is possible using linear relaxation or the "optimal fractional solution" (G. B. Dantzig, 1957; Shih, 1979) to predict some of the items that are likely or unlikely to appear in the good solutions. This study proposes a method for using such prior information as an additional guide for the GA evolutionary process for the MKP01 problem. By guide, we mean any structure external to GA, which maintains its original composition and is used to drive its search process. This can be through a subset of operators, in order to accelerate the search process and improve the speed of convergence. This section aims to describe the GGA components.

6.4.1 Chromosome design

The population is composed of a finite number of chromosomes. A chromosome represents a feasible solution to the problem (MKP01). As mentioned before, the target in the MKP01 is to define the subset of items that maximises the total profit. The GGA chromosome consists of the set of the items to be added to the knapsack. GGA uses the integer representation, where each gene represents an item ID. The items are coded as integer numbers. A chromosome is formed only by the number of items that it contains. This representation allows reducing the size of the processed data (Fig. 6.1).

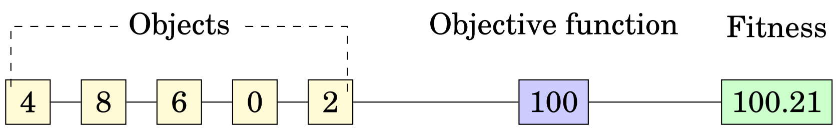

Figure 6.1: Example of the the chromosome design. The objects(items) packed in the knapsack are represented by their identifiers. The objective function value is calculated by summing the benefit of all the objects while the fitness is calculated according to the fitness function.

6.4.2 Guiding Information

The guiding information is based on the work by (Puchinger et al. (2010)). The items are sorted in decreasing order according to a statistical efficiency e_j based on the profit and the cost (e_j^{simple},e_j^{scaled},e_j^{st} or $e_j^{general}$). In simple words, the items are sorted based on how likely each item is to appear in high performing individuals, the item at the top of this list are the items that are likely to be selected while the items at the bottom of the list are the items that are unlikely to appear in good solutions. However, it is important to note here that this list is just an estimate and not a predefined part of the solution. It should be noted also that the Greedy heuristic as by (Senju & Toyoda, 1968) only based on the efficiency sorting is not an effective solution for the strongly correlated problem instances of the MKP01 (Huston et al., 2008).

The sort operation allows favouring items that have a good compromise (i.e. efficiency) between the average profit and overall capacity. The efficiency of an item is high if its profit is high while its required global capacity is low. The sorted items are split into three sets (Fig. 6.2) where the value of each variable is assigned as follows:

- $X_1 : x_j = 1$ The variables have the best efficiency e_j. These variables are most likely to build the best solutions even the optimal solution.
- $Core : x_j =?$ The variables have the values of the efficiency e_j very close. In this group, it is difficult to determine the best.
- $X_0 : x_j = 0$ The variables have a very low efficiency e_j, in other words, the profit is low or the capacity is large or both.

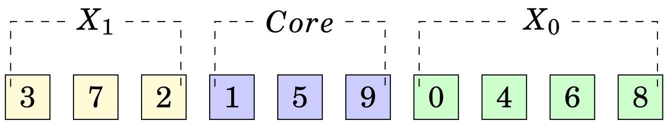

Figure 6.2: Example of the guide construction. The objects (items) are sorted according to the efficiency e_j

The guide is represented by the items of $X_1 \cup Core \cup X_0$. The sizes of X_1, $Core$ and X_0 are determined as follows: Construct a feasible solution by adding the items in the order. The item that makes the solution unfeasible represents the centre of $Core$. The size of each part of the guide depends on the size of $Core$. By determining the size of $Core$, we define the size of the other parts.

6.4.3 Initial population

GGA algorithm uses a special initialisation process which allows the GA to make use of the prior information available about the items, and at the same time generates a diverse initial population to ensure exploration of the search space. A chromosome is generated from the items of X_1 completed by items generated randomly. In each chromosome, X_1 is integrated with a probability α. If α is set to zero this means that all the items in each individual are selected randomly, while $\alpha = 1$ means that each individual in the initial population contains all the items in X_1. This method allows having an initial population of good quality by integrating X_1 and ensures the diversification by adding the rest randomly.

6.4.4 Fitness evaluation

In GGA, the objective function can be different than the fitness function $f(j)$. The first is the value of a solution relative to MKP01 problem. It is evaluated according to the first equation of the MKP01 model (see Section 2.4). While the fitness function is defined in a way to guide the search process of GGA. Different formulations of the fitness function are examined by introducing the efficiency e_j, X_1 and X_0.

1. The fitness function is exactly equal to the objective function (Eq. 6.6) :

$$f(j) = \sum_{j=0}^{n} p_j x_j \tag{6.6}$$

2. The efficiency e_j is introduced in its evaluation according to Eq. 6.7. Each generation, the fitness value of each chromosome is calculated. The fitness formula allows giving more chance to the chromosome that has a high efficiency to be selected more than the others.

$$f(j) = \sum_{j=0}^{n} e_j p_j x_j \tag{6.7}$$

3. X_1 and X_0 are introduced in the fitness measuring; the first as a reward and the second as a penalty (Eq. 6.8). The aim is to reward (respectively penalize) each chromosome according to its similarity with X_1 (respectively X_0). Thus allows, at the same time, increasing the chance for the good chromosome to be selected and decreasing it for the bad ones.

$$f(j) = \sum_{j=0}^{n} p_j x_j + reward - penalty \tag{6.8}$$

   Where $reward = s_1 * p_z$, $penalty = s_0 * p_z$, s_1 and, s_0 represents the similarity rate with X_1 and X_0 respectively, and p_z is a significant percentage of the average objective function of the generation (in the experiments $p_z = 0.1$ is used).

4. The fitness uses the similarity of the chromosome with X_1 as follows:

$$f(j) = (1+s_1)\sum_{j=0}^{n} p_j x_j \tag{6.9}$$

6.4.5 Genetic operators

GGA uses standard genetic crossover and mutation operators. Tournament selection of size 5 is used as the selection method. For the crossover operator, the random single point method is applied with a probability p_c. For mutation, the mutation by random multiple point bit flip is applied with the probability p_m. And finally, a reproduction operator copies a subset of individuals with the probability p_r such as $p_c + p_m + p_r = 1$.

Algorithm 11 The GGA pseudo-code.

```
Require: MKP01 instance
Ensure: a feasible solution $S$
 1: calculate the efficiency $e_j$ for each variable
 2: sort the items according to the efficiency function
 3: calculate $X_1$, $Core$ and $X_0$ of the guide
 4: initialise the population $pop$ with $X_1$ and $\alpha$
 5: for $ctr = 1$ to $ng$ do
 6:     evaluate the fitness for each chromosome in $pop$ according to the fitness
        equation
 7:     crossover with $(pc)$
 8:     mutation with $(pm)$
 9:     reproduction with $(pr)$
10: end for
11: return the best solution $S^*$.
```

6.5 Experimental setup and parameters tuning

This section explains the experimental setup and presents the parameter tuning analysis for the GGA algorithm. It is important to note that the concept of guide can be applied to any problem if an effective method for sorting variables exists. For an experimental purpose, and because the chosen sorting method concerns

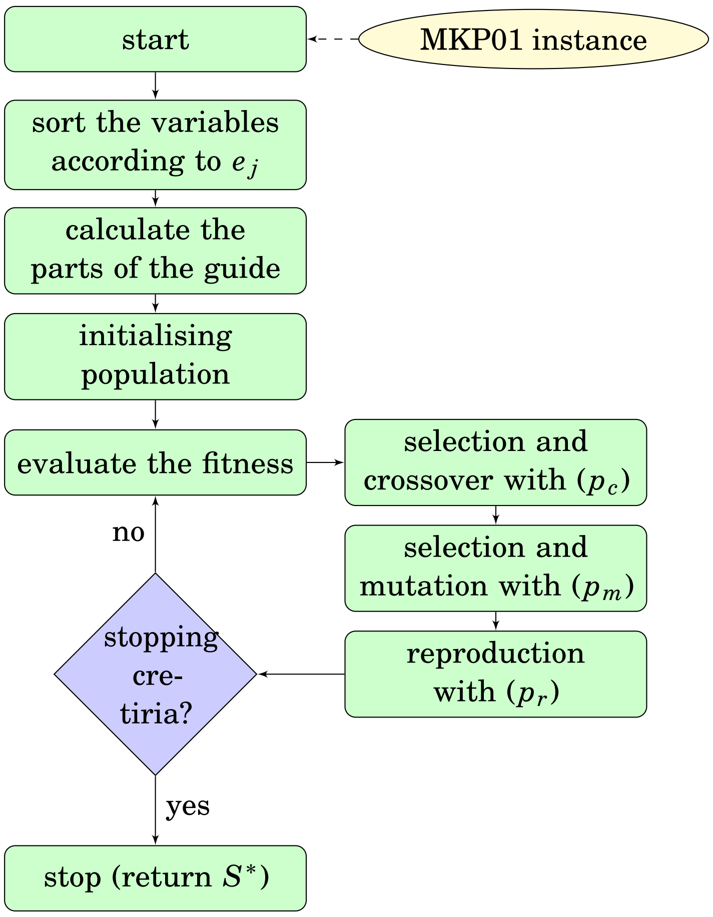
start
MKP01 instance
sort the variables according to e_j
calculate the parts of the guide
initialising population
evaluate the fitness
selection and crossover with (p_c)
selection and mutation with (p_m)
reproduction with (p_r)
stopping cre-tiria?
no
yes
stop (return S^*)

Figure 6.3: Flowchart of the GGA optimisation process.

MKP01, it is natural to use data from this problem. The test platform is a Toshiba laptop with 4GB RAM capacity and an Intel Core (TM) i5-4200 M 2.5 Ghz CPU. The Java language is used to implement the approach. As for the test data, two well-known benchmarks from the OR-Library[1] are used.

Determine the best parameters is a difficult and deterministic task for any algorithm. Unfortunately there is no proven approach to do that. Furthermore, even by using the same parameters, it is not sure to have similar resolution quality for different problems. Generally, the empirical method is the only solution in a similar situation. This method is used in our study. As GGA has several

[1]http://people.brunel.ac.uk/ mastjjb/jeb/orlib/files/

Table 6.1: Parameters of the GGA used to perform the experiments

parameter	description	value
ps	population size	500
ng	number of generations	500
p_c	crossover probability	0.2
p_m	mutation probability	0.7
p_r	reproduction probability	0.1
α	X_1 integration rate on the initial population	0.9
st	selection tour	10
nmp	number of mutation points	3
nbk	number of best chromosomes keeped	5
$nrun$	number of run for each instance	30

parameters, we do a set of experiments aiming to determine the values of the most important parameters. In this sense the subset of instances 5x100-0.25 is used. All parameter values are set as shown in Table 6.1; only the parameter to be measured is changed.

Fig. 6.4 (a), displays how α may affect GGA Distance From the Optimum ($D.F.O$) by changing its value. The study revealed that: α was relatively proportional to $D.F.O$ in the range 0-0.9 and $\alpha = 0.9$ gave the lowest $D.F.O$ and $D.F.O$ value was higher when α exceeded 0.9. This indicates that using more items of I_1 in the initial population individuals improves significantly the quality of the obtained solutions. Furthermore, the integration of the whole group (i.e. $\alpha = 1$) reduces the diversity of individuals.
The Average Distance From the Optimum $A.D.F.O$ of the GGA application on the *OR5x100-0.25* instances with different values of ng is given in Fig. 6.4 (b). The results indicated that when ng was high the $A.D.F.O$ decreased. Moreover, the process kept reaching good solutions and might even reach the optimum for some problem instances. In the next experiments it is considered that $ng = 500$.
The effect of using different values of population size ps on the GGA performance is given in Fig. 6.4 (c). The results showed that more individuals (between 100 and 1000) in the population led to have an $A.D.F.O$ close to the optimum; and $ps = 500$ allowed having the best solutions. $ps = 1000$ gave a low value because a large population requires more time to reach solutions of high-quality. In the

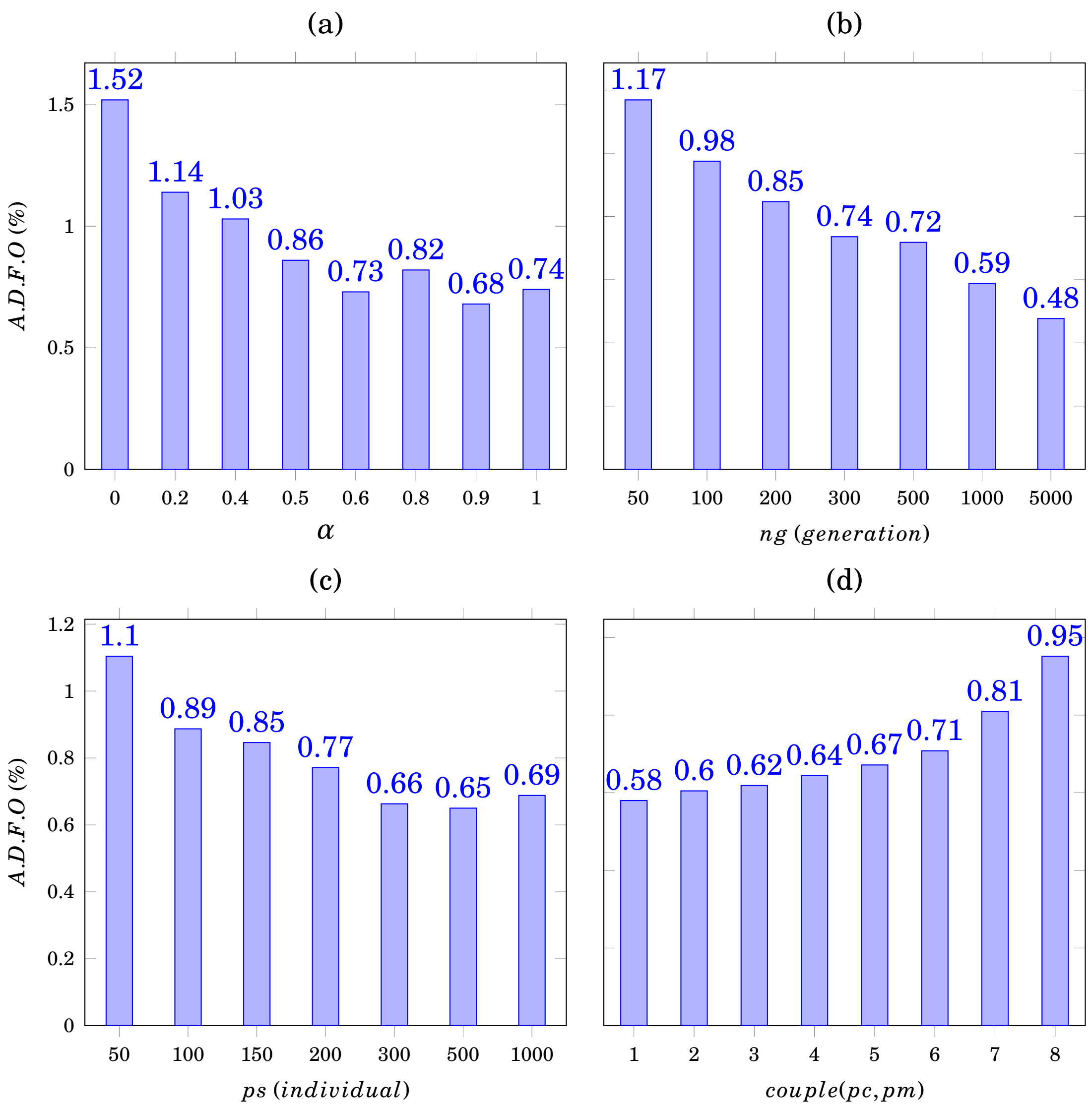

Figure 6.4: Guided-GA tuning of four parameters by calculating the Average Distance From the Optimum $A.D.F.O$ of 30 runs on the OR5x100_0.25 instances. α, number of generation ng, population size ps the couple crossover/mutation probability (pc, pm) are illustrated in (a), (b), (c) and (d) respectively. (pc, pm) value is in {(0.1, 0.8), (0.2, 0.7), (0.3, 0.6), (0.4, 0.5), (0.5, 0.4), (0.6,0.3), (0.7, 0.2), (0.8, 0.1)}

following it is assumed that $ps = 500$.

Eight couple values of crossover and mutation probabilities (p_c, p_m) were compared to identify which one is to be used in GGA. The *OR5x100_0.25* instances

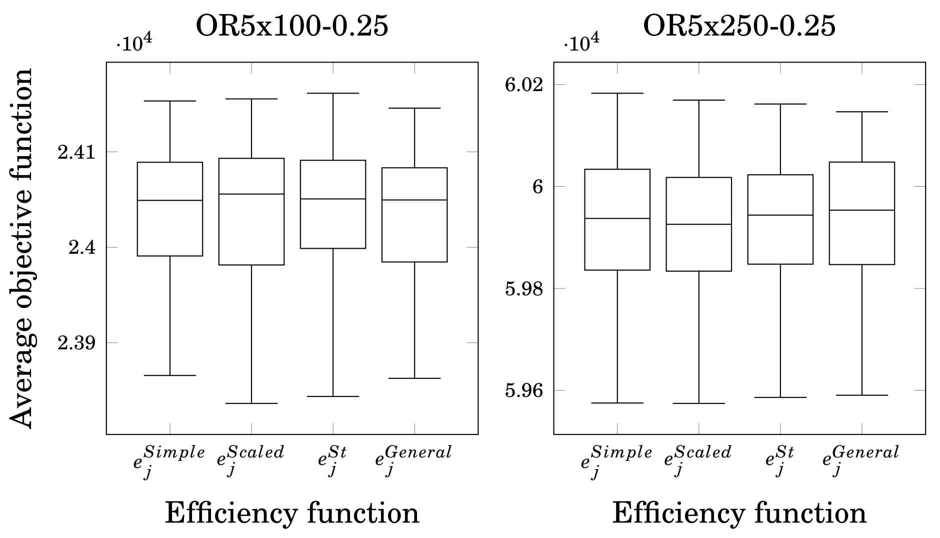

Figure 6.5: Impact of the efficiency measurement. The average objective function values rang obtained by Giuded-GA with different values of efficiency measurement and using the data set *OR5x100-0.25 OR5x250-0.25* and *OR5x500-0.25*.

were applied 30 runs and the $A.D.F.O$ results are shown in Fig 6.4 (d). The $A.D.F.O$ significantly increased when the crossover probability was increased while the mutation probability was decreased (Fig 6.4(d)). The first couple gave the best $A.D.F.O$. This suggests that more mutation and less crossover allows obtaining the best solutions.

A comparison of the efficiency measurement function impact on GGA was investigated. The average objective function range values of its application on the *OR5x250-0.25* and *OR5x500-0.25* instances are illustrated by Fig. 6.5. It was difficult to decide which function is the best.

6.5.1 Core size

This experiment aim to determine the best size of the *Core*. Also, it allows to determine whether enlarging the size I_1 is likely or not to include more items of the optimal solutions. The experiment consists in applying the CPLEX on the *Core* of the 270 instances, and measuring the $A.D.F.O$. this experiment allows also to determine the position of the best items after sorting is performed.

Table 6.2: Average Distance From the Optimum $A.D.F.O$ of the CPLEX application with on the *Core* for different size δ on all the experimental data.

	α	$\delta = 10$	$\delta = 0.1n$	$\delta = 0.15n$	$\delta = 0.2n$
$t = 5$					
5	0.25	14.446	7.589	3.830	1.355
	0.5	5.605	2.885	1.555	0.637
	0.75	1.968	0.826	0.440	0.158
10	0.25	9.777	5.784	3.239	1.679
	0.5	4.555	2.780	1.598	1.055
	0.75	2.127	1.234	0.793	0.393
30	0.25	6.316	4.234	2.817	1.672
	0.5	4.664	3.280	2.264	1.646
	0.75	2.111	1.319	0.832	0.508
$t = 10$					
5	0.25	14.446	7.589	3.830	1.354
	0.5	5.605	2.885	1.555	0.637
	0.75	1.968	0.826	0.440	0.158
10	0.25	9.777	5.784	3.239	1.677
	0.5	4.555	2.780	1.597	1.052
	0.75	2.127	1.234	0.792	0.392
30	0.25	6.316	4.234	2.817	1.662
	0.5	4.664	3.280	2.264	1.642
	0.75	2.111	1.319	0.831	0.507
$t = 50$					
5	0.25	14.446	7.589	3.830	1.350
	0.5	5.605	2.885	1.554	0.636
	0.75	1.968	0.826	0.439	0.158
10	0.25	9.777	5.784	3.239	1.672
	0.5	4.555	2.780	1.596	1.049
	0.75	2.127	1.234	0.792	0.391
30	0.25	6.316	4.234	2.817	1.661
	0.5	4.664	3.280	2.264	1.637
	0.75	2.111	1.319	0.831	0.503
$t = 100$					
5	0.25	14.446	7.589	3.830	1.350
	0.5	5.605	2.885	1.554	0.636
	0.75	1.968	0.826	0.439	0.158
10	0.25	9.777	5.784	3.239	1.669
	0.5	4.555	2.780	1.596	1.048
	0.75	2.127	1.234	0.792	0.391
30	0.25	6.316	4.234	2.817	1.661
	0.5	4.664	3.280	2.264	1.637
	0.75	2.111	1.319	0.831	0.503

Table 6.2 summarises the obtained $A.D.F.O$ with $\delta = \{10, 0.1n, 0.15n, 0.2n\}$ with n is the number of items. The results indicated that the larger is δ the better is the $A.D.F.O$. Also, it revealed that the optimal solution, after the sort, was probably gathered in a subset of 20% of the items. This could means, in our case to include more items from the *Core* in I_1.

Table 6.3: Comparison of the $D.F.O$ obtained by GGA with different expression of the $fitnessfunction$, using the *OR5x100-0.25* dataset

instance	Eq.6.6	Eq.6.7	Eq.6.8	Eq.6.9
1	1,180	**0,682**	1,318	1,017
2	1,237	**0,757**	1,111	0,908
3	0,843	**0,382**	0,781	0,487
4	1,254	**0,663**	1,488	0,967
5	0,951	**0,583**	0,955	0,768
6	1,061	**0,523**	1,084	0,666
7	1,710	**0,856**	2,111	1,485
8	1,414	**0,570**	1,249	0,949
9	1,086	**0,620**	0,909	0,813
10	1,230	**0,563**	1,112	0,745

6.5.2 Comparing the fitness functions

A comparison between the four expressions of fitness function is reported in Table 6.3. It revealed that the use of e_j to express the fitness function is the best choice. The $A.D.F.O$ that showed the stability of the obtained results suggest that e_j expresses the guidance in the best way. The equation increases the chance of individuals having the most good variables (in terms of e_j) to be selected, in parallel, it reduces it for those having more bad variables. Other expressions may give better results as assigning a decrease weight to the variables according to the order, but it is still true that the idea of expressing the values of fitness with e_j is interesting.

6.6 Simulation results

In order to provide a comprehensive analysis of the proposed GGA algorithm, this section provides two sets of experiments. The first set of experiments aims to evaluate the performance gain from adding the "guiding" component to the standard GA supported with statistical analysis. This is achieved by comparing the GGA against GA using the same evolutionary parameters (Section 6.6.1). The second set of experiments aims to compare the proposed GGA with the state-of-art results reported in the literature (Section 6.6.2).

6.6.1 Comparing GGA to GA

A comparison between GA and GGA was conducted to measure the contribution of the data pre-analysis information on the convergence of GA. Both GGA and GA were executed 30 times during 200 generation on the *OR5x100-0.25_1*. The obtained objective function value of each generation was recorded. The average objective function of both approaches is compared in Fig. 6.6. The curves indicated two search steps: diversification (0-35) and intensification (36-200) in both GA-Guided kept a large gap on GA and maintained it throughout the process.
An extended investigation was done using the first and the last instances of each class of the Chu & Beasley benchmarks. The comparison was done including many factors: Average $D.F.O$, Best $D.F.O$, Worst $D.F.O$ and average processing $Time$ (Table 6.4). It was noticed that GGA obtained better solutions than GA in most instances. GGA was able to reach the optimal solution for some instances. Also, for the instances with $\alpha = 0.50$ and 0.75, both approaches were closer to the optimum than with $\alpha = 0.25$.
Moreover, to analyse the capacity of GGA to obtain solutions of good quality, the range of objective function value of 30 runs compared to GA on six instances is given in Fig. 6.7. This illustration suggested that through 30 runs, the quality of solutions that GGA was able to reach was largely improved than GA.
It could be concluded that the information about the addressed problem known in advance when integrated for guiding the GA process, improves its convergence capacity and enhances the quality of the results.

6.6.1.1 Statistical analysis

An ANalysis Of the Variance (ANOVA) pairwise comparison was conducted to investigate whether or not the guidance has a real effect on the GA. It has been supposed that the null hypothesis H_0 is "GGA has not a significant improvement on GA". The first comparison included only one instance (Table 6.6 and 6.7) while

Table 6.4: The GGA compared to GA in term of *A.D.F.O*, Best *D.F.O*, Worst *D.F.O* and Average exection *Time*. The first and the last instance of each group is used

		GGA				GA			
	Instance	*Average D.F.O*	*Best D.F.O*	*Worst D.F.O*	*Time (sec)*	*Average D.F.O*	*Best D.F.O*	*Worst D.F.O*	*Time (sec)*
5x100	0.25_01	**0.56**	**0**	**1.13**	3	2.46	1.45	4.71	3
	0.25_10	**0.58**	**0**	**1.18**	3	1.55	0.55	2.71	2
	0.50_01	1.03	0.73	**1.6**	5	**0.74**	**0.12**	1.73	5
	0.50_10	**0.44**	**0.22**	**0.77**	5	0.91	0.43	1.79	4
	0.75_01	0.44	**0**	0.78	7	**0.35**	0.12	**0.58**	6
	0.75_10	0.57	0.21	**0.74**	7	**0.34**	**0.09**	0.77	6
5x250	0.25_01	**0.83**	**0.42**	**1.25**	7	3.45	2.68	5.32	5
	0.25_10	**0.8**	**0.25**	**1.51**	7	4.33	2.74	5.63	4
	0.50_01	**0.53**	**0.31**	**0.73**	13	1.21	0.71	1.86	11
	0.50_10	**0.59**	**0.38**	**0.93**	12	0.97	0.48	1.47	10
	0.75_01	**0.4**	**0.18**	**0.62**	82	0.53	0.29	0.78	17
	0.75_10	**0.39**	**0.24**	**0.59**	19	0.44	0.24	0.85	17
5x500	0.25_01	**0.87**	**0.52**	**1.32**	15	4.21	3.46	5.26	10
	0.25_10	**0.91**	**0.58**	**1.57**	16	3.82	3.14	4.74	11
	0.50_01	**0.42**	**0.23**	**0.81**	29	1.58	1.23	1.94	23
	0.50_10	**0.5**	**0.29**	**0.73**	29	1.31	0.9	1.74	24
	0.75_01	**0.25**	**0.14**	**0.41**	47	0.64	0.44	0.86	41
	0.75_10	**0.3**	**0.14**	**0.55**	51	0.59	0.42	0.9	40
10x100	0.25_01	**0.81**	**0.2**	**1.88**	3	1.77	1.31	2.63	3
	0.25_10	**1.02**	**0**	**1.82**	3	2.82	1.29	4.63	3
	0.50_01	**0.67**	**0.24**	**1.21**	5	1.2	0.51	2.09	5
	0.50_10	**0.47**	**0.27**	**0.65**	6	1.87	0.7	2.88	5
	0.75_01	**0.31**	**0.23**	**0.44**	8	0.51	0.25	1.4	7
	0.75_10	**0.09**	**0**	**0.32**	8	0.61	0.3	0.94	7
10x250	0.25_01	**0.92**	**0.39**	**1.56**	9	3.76	2.16	4.93	6
	0.25_10	**0.88**	**0.61**	**1.59**	9	3.65	2.78	4.41	7
	0.50_01	**0.56**	**0.26**	**0.84**	15	1.29	0.93	1.9	12
	0.50_10	**0.49**	**0.23**	**0.95**	16	1.7	0.85	2.78	13
	0.75_01	**0.28**	**0.14**	**0.52**	23	0.54	0.29	0.95	19
	0.75_10	**0.29**	**0.08**	**0.6**	23	0.69	0.36	1.08	19
10x500	0.25_01	**1.01**	**0.58**	**1.62**	19	2.78	2.17	4.05	14
	0.25_10	**0.88**	**0.57**	**1.24**	19	3.46	2.38	4.62	13
	0.50_01	**0.46**	**0.28**	**0.84**	35	1.45	1.04	2.15	28
	0.50_10	**0.54**	**0.32**	**0.8**	35	1.49	1.07	2.14	27
	0.75_01	**0.26**	**0.12**	**0.41**	53	0.8	0.63	1.11	44
	0.75_10	**0.26**	**0.13**	**0.59**	53	0.68	0.52	0.82	45
30x100	0.25_01	**1.58**	**0.79**	**2.73**	3	2.15	1.08	3.73	4
	0.25_10	**1.28**	**0.18**	**2.63**	4	2.15	1.6	4.7	3
	0.50_01	**1.18**	**0.54**	**1.79**	7	1.76	0.99	2.13	6
	0.50_10	**0.66**	**0.42**	**1.17**	7	2.5	1.61	3.19	5
	0.75_01	**0.48**	**0.15**	1.23	10	0.9	0.46	**0.97**	8
	0.75_10	**0.35**	**0.11**	**0.6**	10	0.54	0.34	0.93	9
30x250	0.25_01	**2.09**	**1.83**	**3.13**	9	2.88	1.86	3.93	9
	0.25_10	**2.68**	**1.51**	**3.29**	8	3.61	2.6	4.79	8
	0.50_01	**1.17**	0.74	**1.58**	18	1.23	**0.61**	2.41	18
	0.50_10	**1.2**	**1.04**	**1.41**	18	1.63	1.35	2.12	17
	0.75_01	**0.38**	**0.23**	**0.64**	30	0.9	0.68	0.99	25
	0.75_10	**0.58**	**0.41**	**0.77**	30	0.77	0.5	1.29	25
30x500	0.25_01	**3.39**	**2.18**	**4.69**	100	4.83	3.48	5.49	164
	0.25_10	3.9	**2.9**	6.29	99	**3.58**	3.37	**3.89**	16
	0.50_01	2.2	2.05	2.42	35	**1.45**	**1.36**	**1.72**	33
	0.50_10	**1.51**	**1.25**	**1.77**	41	1.63	1.48	2.39	35
	0.75_01	**0.41**	**0.35**	**0.5**	67	0.68	0.41	1.01	61
	0.75_10	**0.35**	**0.26**	**0.49**	77	0.6	0.47	1	59

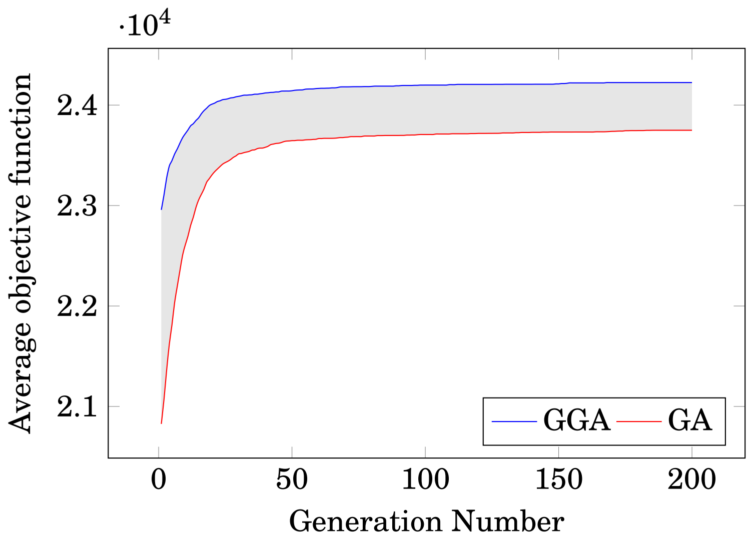

Figure 6.6: Comparing the convergence of GGA with GA using the *OR5x100-0.25_1* instance

the second included all the instances (Table 6.8 and 6.9). The first comparison indicated an $F = 152.72$ largely greater than $Fcrit = 3.99$, and a $P-value = 2.23E-18$ largely lower than $\alpha = 0.05$. The second comparison including all the instances showed an $F = 9.58$ greater than $F\ crit = 4.03$, and a $P-value = 0.003$ lower than $\alpha = 0.05$. Both results confirm that GA is significantly improved by adding the guidance to its search process. Consequently, the null hypothesis can be rejected.

A t-Test, ANOVA and t-Test (with Two-Sample Assuming Unequal Variances) comparative analysis between GGA and the approaches reported in section 6.6.1 was conducted. The same results have been used to perform the comparison reported in Table 6.10. The obtained t-Test values of the pairwise GGA comparison to the other approaches were less than the $P-value = 0.05$ except SAHS-SLS. Also the ANOVA comparison results indicated $P-value$ less than $\alpha = 0.05$ and F largely greater than $F\ crit$ except when compared with SAHS-SLS. The t-Test (with Two-Sample Assuming Unequal Variances) indicated that the $tStat$ was negative and a $P-value$ less than the $P-value$ (i.e. $\alpha = 0.05$) associated with

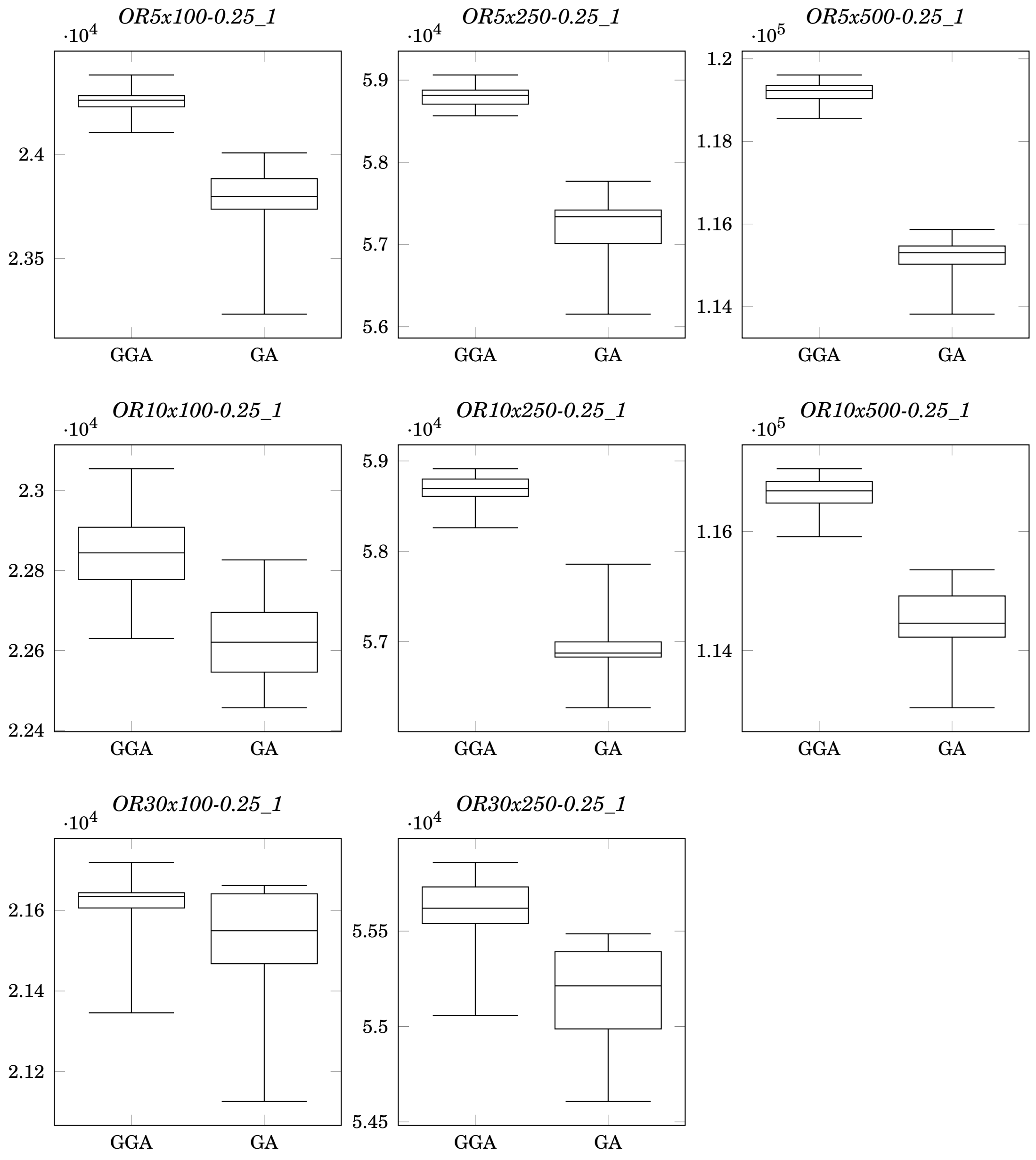

Figure 6.7: The objective function values rang obtained by Guided-GA compared with GA within 30 run

that in both one-tail level and two-tail level excluding the GGA vs. SAHS-SLS comparison. From all this statistical analysis it may be concluded that the null hypothesis H_0 is rejected. Therefore, GGA performs significantly better than the other approaches and is comparative to the SAHS-SLS.

Table 6.5: Comparison of GGA to GA, SAHS-SLS, SGHS and HS in terms of $A.D.F.O$ and average $Time$

	GGA		GA		SAHS-SLS		SGHS		HS		Greedy
Data-set	$D.F.O$	$time$	$D.F.O$	$time$	$D.F.O$	$time$	$D.F.O$	$time$	$D.F.O$	$time$	$D.F.O$
5x100 0.25	0.62	3	2.17	2.2	**0.60**	18.9	0.78	16.3	5.36	9.5	29.25
0.50	0.67	5.1	0.86	4.0	**0.57**	24.8	0.78	22.3	5.15	12.4	13.01
0.75	0.34	6.9	0.42	6.0	**0.32**	29.1	0.47	27.6	3.09	16.6	6.40
Average	0.54	5	1.15	4.1	**0.49**	24.2	0.68	22.1	4.53	12.8	16.22
5x250 0.25	**0.80**	7.6	4.03	4.8	2.14	31.5	2.62	26.6	8.41	22.4	23.81
0.50	**0.56**	12.6	1.15	10.4	1.48	68.1	2.27	57.2	6.21	36.3	9.10
0.75	**0.33**	18.9	0.58	16.8	1.22	99.3	1.77	80.5	3.78	62.3	4.07
Average	**0.56**	13.0	1.92	10.7	1.61	66.3	2.22	54.7	6.13	40.3	12.33
5x500 0.25	**0.84**	15.6	4.27	10.5	1.85	146.3	3.75	109.1	9.57	66.7	20.70
0.50	**0.53**	30.8	1.45	24.0	1.31	248.4	3.60	177.3	7.25	116.4	9.62
0.75	**0.26**	47.7	0.65	42.3	1.33	340.0	2.69	269.6	4.67	187.6	3.39
Average	**0.54**	31.4	2.12	25.6	1.50	244.9	3.34	185.3	7.16	123.6	11.24
10x100 0.25	1.23	3.0	2.40	3.0	**0.90**	29.0	1.26	27.8	5.31	16.5	21.69
0.50	**0.61**	5.2	1.53	5.0	0.64	46.6	0.97	33.6	4.90	20.6	10.20
0.75	**0.35**	7.9	0.53	6.8	0.36	48.8	0.64	43.5	3.22	27.3	4.91
Average	0.73	5.4	1.49	4.9	**0.63**	41.5	0.96	35.0	4.48	21.5	12.26
10x250 0.25	**0.98**	8.9	3.56	6.2	1.49	93.3	2.33	74.1	7.17	44.7	15.94
0.50	**0.58**	15.0	1.35	12.6	0.91	160.2	2.06	116.3	6.01	76.6	7.91
0.75	**0.33**	23.0	0.66	19.2	0.67	261.5	1.59	172.8	3.85	116.6	3.57
Average	**0.63**	15.6	1.86	12.7	1.03	171.7	1.99	121.1	5.68	79.3	9.14
10x500 0.25	**0.91**	18.5	3.61	13.2	2.11	157.1	4.16	118.5	9.95	78.0	15.39
0.50	**0.50**	32.6	1.44	27.4	1.61	267.0	4.23	207.4	7.95	143.5	6.10
0.75	**0.31**	50.3	0.71	45.4	1.45	384.4	4.11	297.3	5.04	229.3	2.61
Average	**0.57**	33.8	1.92	28.7	1.72	269.5	4.16	207.7	7.65	150.3	8.03
30x100 0.25	1.66	3.6	2.27	3.0	**1.23**	77.0	0.93	68.5	2.79	49.7	18.04
0.50	1.10	6.0	1.72	5.8	**0.68**	64.9	0.96	52.5	3.90	36.5	8.83
0.75	0.49	9.6	0.78	8.9	**0.33**	87.1	0.52	67.9	3.22	46.8	5.97
Average	1.08	6.4	1.59	5.9	**0.75**	76.3	0.80	63.0	3.30	44.3	10.94
30x250 0.25	2.03	9.2	3.20	8.6	**1.93**	155.9	2.47	91.9	7.59	61.8	11.53
0.50	1.17	17.1	1.46	17.3	**0.84**	155.8	2.49	103.0	7.00	71.6	5.55
0.75	**0.48**	27.7	0.73	28.3	0.66	231.7	1.77	150.7	4.28	108.3	3.33
Average	1.23	18.0	1.80	18.1	**1.14**	181.1	2.25	115.2	6.29	80.6	6.80
30x500 0.25	4.07	99.1	3.50	18.8	**2.24**	173.3	5.63	127.1	9.71	89.0	44.18
0.50	1.90	40.4	1.45	36.1	**1.19**	264.2	4.82	206.3	8.39	137.1	6.21
0.75	**0.43**	63.4	0.69	58.5	0.87	344.0	6.31	285.1	6.44	243.3	2.30
Average	2.14	67.6	1.88	37.8	**1.43**	260.5	5.58	206.2	8.18	156.4	17.56

6.6.2 Comparison with the literature

As with most optimisation problems, MKP01 heuristics could be classified into two groups: the first is*constructive* heuristics, that aim to construct a solution. The second is *improvement* heuristics which aim to improve a given initial solution generated first by a *constructive* heuristic. The proposed method is considered as a

Table 6.6: Statistic summary of GGA and GA on *OR100x5-0.25_1*

Algorithm	*Count*	*Sum*	*Average*	*Variance*	*Std. Dev*	*Best*	*Worst*
GGA	32	20.53	0.64	0.12	0.34	0.30	0.98
GA	32	82.88	2.59	0.68	0.82	1.77	3.41

Table 6.7: ANOVA comparison between GGA and GA on *OR100x5-0.25_1*

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	60.75	1	60.75	152.75	2.23E-18	3.996
Within Groups	24.66	62	0.40			
Total	85.40	63				

Table 6.8: Statistic summary of GGA and GA on Chu & Beasley instances

Algorithm	*Count*	*Sum*	*Average*	*Variance*	*Std. Dev*	*Best*	*Worst*
GGA	27	24.08	0.89	0.63	0.80	1.69	0.10
GA	27	47.19	1.75	1.43	1.20	2.94	0.55

Table 6.9: ANOVA comparison between GGA and GA on the Chu & Beasley instances

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	9.89	1	9.89	9.58	0.003	4.03
Within Groups	53.67	52	1.03			
Total	63.56	53				

Table 6.10: Statistical comparison using t-Test, ANOVA and Two-Sample Assuming Unequal Variances of GGA to Greedy, GA SAHS, SGHS and HS through the Chu & Beasley instances

	GGA vs. Greedy	**GGA vs. GA**	**GGA vs. SAHS**	**GGA vs. SGHS**	**GGA vs. HS**
t-Test	2.3E-06	0.002	0.093	3.7E-05	2.7E-13
P-value	5.1E-07	0.003	0.19	4.6E-05	9.1E-16
F	32.85	9.58	1.80	19.75	130.08
F crit	4.03	4.03	4.03	4.03	4.03
t Stat	-5.54	-2.89	-1.36	-4.50	-11.02
P(T<=t) one-tail	4.6E-06	0.003	0.091	3.2E-05	1E-12
t Critical one-tail	1.71	1.68	1.68	1.69	1.69
P(T<=t) two-tail	9.3E-06	0.006	0.182	6.4E-05	2E-12
t Critical two-tail	2.06	2.02	2.01	2.03	2.04

constructive heuristic. However, in order to demonstrate the performance of the proposed method, the performance of the GGA is compared with both *constrictive* and *improvement* approaches. The following text gives a short description of the methods (*constructive* and *improvement*) used in the comparison presented in this section.

Constructive: PECH (Primal Effective Capacity Heuristic) (Akçay et al., 2007) is a simple greedy heuristic which incorporates a strategy based on the *effective capacity* for selecting and adding the items to the knapsack. Thus, PECH is a polynomial-time algorithm with the computational complexity $O(mn^2)$. Magazine And Oguz (MAG) (Magazine & Oguz, 1984) proposed an algorithm based on the Lagrange multipliers approach. The multipliers and variables values are initialised to zero and one respectively. Iteratively, the approach increases the corresponding multiplier to the largest remain/total capacity ratio until a certain constraint is violated and then set a variable to zero. Volgenant and Zoon (VZ) (Volgenant & Zoon, 1990) improved this heuristic by computing more than one multiplier value in each iteration and readjusting these values at the end of the algorithm. PIR (Pirkul, 1987) is an approach based on a dual surrogate relaxation of the 0-1 MKP01 supported with a branch and bound method. The proposed solution significantly reduces the solution times compared to the resolution of the initial problem. SCE (Shuffled Complex Evolution) is applied for MKP01 in (Baroni & Varejão, 2015). It is a population based meta-heuristic which shuffles the population into complexes according to the fitness ascending sort. Iteratively, different crossing alternatives and shuffling are operated within the complexes during the optimisation process.

Improvement: CB (Chu & Beasley, 1998) is a standard genetic algorithm augmented with a feasibility and constraint operator which utilises problem-specific knowledge. The proposed GA obtains high-quality solutions, however, it demands a long computational time. In a purpose of reducing the problem size and simplify its resolving, NR (P) (New Reduction (Pirkul)) (Hill et al., 2012), operates a lagrangian dual relaxation on MKP01, and proposes a dynamic estimation of the core problem size relative to the problem difficulty. The core is then solved with a greedy heuristic combined with a local improvement phase. MCF (Modified Choice Function - Late Acceptance Strategy) (Drake et al., 2015) is a hyper-heuristic based on heuristics selection function "Modified Choice Function". A score F_t for each low-level heuristic h_j is calculated which includes its dependence with the other low-level heuristics, its previous performance and an elapse time of selection.

GGA is compared with the standard GA algorithm and other state-of-the-art optimisation methods reported in the literature. The comparison is shown in Table 6.11 and Fig. 6.8. As shown in table 6.11, GGA is competitive with both construction and improvement methods and has managed to outperform both group of methods on a few instances.

6.7 Conclusion

The aim of this chapter was to introduce a new hybrid algorithm named Guided Genetic Algorithm (GGA) for solving the Multidimensional Knapsack Problem (MKP01). GGA is a two-step approach that analyses the problem data based on a greedy method in the first step. Useful information about the items of the MKP01, extracted in the first step, are integrated in the second step as a guide in the initialisation and evaluation operators of a GA. To validate the approach, several experiments were conducted on well-known MKP01 test data. The research has shown that adding the guidance has significantly improved the performance of the GA and accelerated its processing time. The statistical experiments confirmed that GGA has a real impact on GA performance. One of the most significant findings

Table 6.11: Comparison of results obtained by GGA with GA, constructive and improvement heuristics

					Constructive					Improvement		
n	m	α	GGA	GA	PECH	MAG	VZ	PIR	SCE	CB	NR(P)	MCF
5	100	0.25	**0.62***	2.17	7.3	13.6	10.3	1.6	3.5	0.99	0.94	1.09
		0.50	**0.66**	0.86	3.4	6.7	6.9	0.77	2.6	0.45	**0.44***	0.57
		0.75	**0.34**	0.42	2.02	5.1	5.6	0.48	1.1	0.32	**0.22***	0.38
	250	0.25	0.79	4.03	7.1	6.6	5.8	**0.53**	4.3	**0.23***	0.46	0.41
		0.50	0.56	1.15	3.2	5.2	4.4	**0.24**	3.3	**0.12***	0.17	0.22
		0.75	0.33	0.58	1.8	3.5	3.5	**0.16**	1.5	**0.08***	0.1	0.14
	500	0.25	0.83	4.27	6.4	4.9	4.1	**0.22**	4.6	1.56	**0.15***	0.21
		0.50	0.53	1.45	3.4	2.9	2.5	**0.08**	3.6	0.79	**0.06***	0.1
		0.75	0.26	0.65	1.7	2.3	2.41	**0.06**	1.8	0.48	**0.03***	0.06
10	100	0.25	**1.23**	2.40	8.2	15.8	15.5	3.4	6.8	**0.09***	2.05	1.87
		0.50	**0.61**	1.53	3.7	10.4	10.7	1.8	5.1	**0.04***	0.81	0.95
		0.75	**0.35**	0.53	1.8	6.1	5.67	1.1	2.4	**0.03***	0.44	0.53
	250	0.25	**0.98**	3.56	5.8	11.7	10.5	1.1	6.9	**0.51***	0.88	0.79
		0.50	0.58	1.35	2.5	6.8	5.9	**0.57**	5.4	**0.25***	0.39	0.41
		0.75	**0.32**	0.66	1.5	4.4	3.7	0.33	2.8	**0.15***	0.19	0.24
	500	0.25	0.9	3.61	5.1	8.8	7.9	**0.52**	6.8	**0.24***	0.34	0.44
		0.50	0.5	1.44	2.4	5.7	4.1	**0.22**	5.8	**0.11***	0.14	0.2
		0.75	0.31	0.71	1.2	3.6	2.9	**0.14**	3.4	**0.07***	0.1	0.13
30	100	0.25	**1.65***	2.27	6.8	17.3	17.2	9.1	8.6	2.91	2.24	3.61
		0.50	**1.09***	1.72	3.2	11.8	10.1	3.51	6.6	1.34	1.32	1.6
		0.75	**0.49***	0.78	1.9	6.58	5.9	2.03	3.6	0.83	0.8	0.97
	250	0.25	**2.03**	3.20	4.8	13.5	12.4	3.7	8.3	**1.19***	1.27	1.75
		0.50	**1.16**	1.46	2.1	8.6	7.1	1.5	6.9	**0.53***	0.75	0.79
		0.75	**0.48**	0.73	1.2	4.4	3.9	0.84	3.8	**0.31***	0.38	0.43
	500	0.25	4.07	3.50	3.7	9.8	9.6	**1.89**	8.6	**0.61***	0.89	1.05
		0.50	1.9	1.45	1.7	7.1	5.7	**0.73**	7.4	**0.26***	0.36	0.44
		0.75	0.43	0.69	0.9	3.7	3.5	**0.48**	4	**0.17***	0.23	0.27

of this study is that prior-knowledge about the data of a CO problem could be significantly helpful to accelerate its solving.

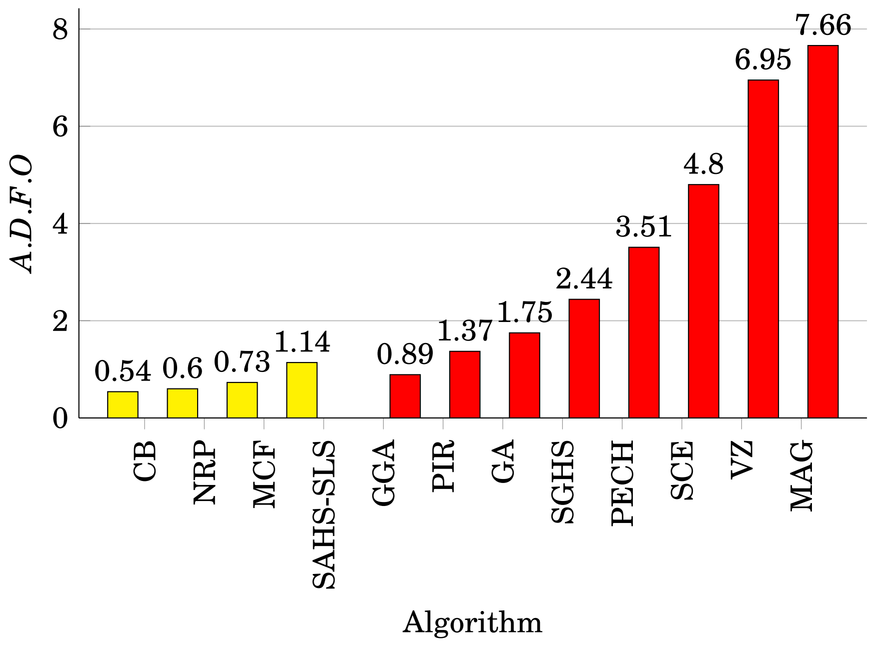

Figure 6.8: Comparing the average distance from the optimum $A.D.F.O$ obtained by GGA using the Chu & Beasley instances with nine other algorithms from the literature. The *improvement* methods are

Chapter 7

Conclusions

Combinatorial optimisation (CO) is important for a wide range of scientific and industrial processes. In particular, the 0-1 Multidimensional Knapsack Problem (MKP01) is one of the most studied CO problems thanks to its many applications in different fields. This project was undertaken for designing approximate solutions that deal with the MKP01. The main goal was to propose hybrid heuristics able to return solutions of high quality within a reasonable time. The second aim of this study was to evaluate the proposed approaches on well-known MKP01 benchmarks and investigate their effects on state-of-art methods.

Three categories of solutions have been suggested based on classic heuristics. The first (Chapter 4) consists in utilising multiple local search methods to solve MKP01 and also the winner determination problem in multi-unit combinatorial auction (MU-WDP) which is an application of MKP01. The second (Chapter 5) hybridises population-based and local search algorithms for ensuring diversification and intensification of a developed approaches. The last (Chapter 6) combines greedy and evolutionary algorithms in a two-step approach in such a way to exploit prior-knowledge about optimal solutions, for improving the genetic algorithm performance.

This study has shown that the local search algorithms (SA, SLS, SLSA, TS and HC) employed for solving both MKP01 and MU-WDP performed successfully on simple and small samples of data. According to the experimental studies, SLS and SLSA outperformed the other methods with encouraging results. The research has also revealed that the hybrid methods (MSA, GA-SLS, GA-SA and SAHS-SLS) had better capability on large benchmarks. The conducted experiments confirmed that the proposed approaches were competitive to the state-of-the-art approaches. This study has also found that, generally, prior knowledge about the structure of final solutions, even if partial, could dramatically simplify its solving. Moreover, when GA was augmented by integrating prior information about MKP01, GA's processing time was reduced and better results have been achieved.

Taken together, these findings suggest a role for our proposed hybrid methods in promoting the MKP01 solving in particular and the CO domain in general. Although this study focuses on solving MKP01, the findings may well have a bearing on other problems.

This work extends our knowledge of solving the CO problems, in particular np-hard ones. It will serve as a base for future studies on MKP01. The content of this thesis makes several contributions to the current literature. First, the literature review gathered and classified the main methods in literature and focused on the hybrid heuristics. It could be a source for the researches of CO domain, especially those studying MKP01. Second, too much work has been undertaken on local search methods where new methods were suggested. These are applicable to other CO problems; they may even be applied on real applications. Third, the present study makes several noteworthy contributions in terms of hybrid approaches that were proven a high performance on complex data and will do similarly on other problems. Moreover, the key strengths of this study are its diversified heuristics and the huge amount of experimental study, but also the number of proposed approaches.

The main weakness of this study was the lack of data for the MU-WDP. Besides, being limited to MKP01, this study lacks more validation on other problems and real applications. An issue that was not addressed in this study was whether hybridising heuristics with exact method gives better options. Additionally, it is unfortunate that the study did not include recent data, therefore the used sample could have affected the generalisability of these results.

This research has thrown up many questions in need of further investigation. More research is required to determine the efficacy of the proposed methods. In terms of directions for future research, it would be interesting to try to involve some of the techniques presented in this thesis in global selective hyper-heuristics. Additionally, combining our algorithms with deterministic methods such as Branch and Cut, Branch and Bound and so on, is likely to produce encouraging outcomes. Also, further experiments on recent benchmarks, is probably recommended to validate the findings of this work.

Appendix A

Publications & Conference papers

Journal articles

2016 Rezoug, A., & Boughaci, D. (2016). A self-adaptive harmony search combined with a stochastic local search for the 0-1 multidimensional knapsack problem. *International Journal of Bio-Inspired Computation* 8.4 pp.234-239.

2017 Rezoug, A., Badr-El-Den, M., & Boughaci, D. (2017). Guided Genetic Algorithm for the 0-1 Multidimensional Knapsack Problem. *Memetic Computing* pp.1-14.

Conference papers

2014 Rezoug, A., & Boughaci, D. (2014). Local Search Methods for the Winner Determination Problem in Multi-Unit Combinatorial Auctions. In *Proceedings of the Third International Conference on Soft Computing for Problem Solving.* pp.589-599. Springer India.

2015 Rezoug, A., Boughaci, D., & Rezoug, A. (2015). Stochastic local search combined with simulated annealing for the 0-1 multidimensional knapsack problem. In *Symposium on Complex Systems and Intelligent Computing (CompSIC)*. Souk-Ahrass, Algeria.

2015 Rezoug, A., Boughaci, D., & Badr-El-Den, M. (2015, September). Memetic algorithm for solving the 0-1 multidimensional knapsack problem. In *Portuguese Conference on Artificial Intelligence* (pp. 298-304). Springer International Publishing.

2016 Rezoug, A., Badr-El-Den, M., & Boughaci, D. (2016, September). Problem Knowledge Information to Improve the Performance of the Genetic Algorithm. Oral communication In *the Operational Research Annual Conference (OR58)*, Portsmouth, United-Kingdom.

2016 Rezoug, A., Badr-El-Den, M., & Boughaci, D. (2016, October). Genetic Algorithm Guided By Pretreatment Information For The 0-1 Multidimensional Knapsack Problem. *International Conference on Metaheuristics and Nature Inspired Computing (META 2016)*, Marrakech, Morocco.

2016 Rezoug, A., Boughaci, D., & Badr-El-Den, M. (2016, October). Solving the 0-1 Multidimensional Knapsack Problem Using a Memetic Search Algorithm. *International Conference on Metaheuristics and Nature Inspired Computing (META 2016)*, Marrakech, Morocco.

2017 Rezoug, A., Badr-El-Den, M., & Boughaci, D. (2017). Knowledge-Based Genetic Algorithm for the 0-1 Multidimensional Knapsack Problem. *IEEE Congress on Evolutionary Computation CEC-2017*, june, 2017. San Sebastien, Spain.

References

Aboudi, R., & Jörnsten, K. (1994). Tabu search for general zero-one integer programs using the pivot and complement heuristic. *ORSA Journal on Computing*, *6*(1), 82–93.

Acan, A., & Tekol, Y. (2003). Chromosome reuse in genetic algorithms. In *Genetic and evolutionary computation conference* (pp. 695–705).

Ahrens, J., & Finke, G. (1975). Merging and sorting applied to the zero-one knapsack problem. *Operations Research*, *23*(6), 1099–1109.

Akbar, M. M., Rahman, M. S., Kaykobad, M., Manning, E. G., & Shoja, G. C. (2006). Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. *Computers & operations research*, *33*(5), 1259–1273.

Akçay, Y., Li, H., & Xu, S. H. (2007). Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research*, *150*(1), 17–29.

Alaya, I., Solnon, C., & Ghédira, K. (2004). Ant algorithm for the multi-dimensional knapsack problem. In *International conference on bioinspired optimization methods and their applications (bioma 2004.*

Alonso, C. L., Caro, F., & Montaña, J. L. (2005). A flipping local search genetic algorithm for the multidimensional 0-1 knapsack problem. In *Conference of the spanish association for artificial intelligence* (pp. 21–30).

Andersson, A., Tenhunen, M., & Ygge, F. (2000). Integer programming for combinatorial auction winner determination. In *Multiagent systems, 2000. proceedings. fourth international conference on* (pp. 39–46).

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., & Protasi, M. (2012). *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media.

Balas, E. (1965). An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, *13*(4), 517–546.

Balas, E. (1967). Discrete programming by the filter method. *Operations Research*, *15*(5), 915–957.

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

Balas, E., & Zemel, E. (1980). An algorithm for large zero-one knapsack problems. *operations Research*, *28*(5), 1130–1154.

Balev, S., Yanev, N., Fréville, A., & Andonov, R. (2008). A dynamic programming based reduction procedure for the multidimensional 0–1 knapsack problem. *European Journal of Operational Research*, *186*(1), 63–76.

Baroni, M. D. V., & Varejão, F. M. (2015). A shuffled complex evolution algorithm for the multidimensional knapsack problem. In *Iberoamerican congress on pattern recognition* (pp. 768–775).

Battiti, R., & Tecchiolli, G. (1992). Parallel biased search for combinatorial optimization: genetic algorithms and tabu. *Microprocessors and Microsystems*, *16*(7), 351–367.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, *6*(2), 154–160.

Beaujon, G. J., Marin, S. P., & McDonald, G. C. (2001). Balancing and optimizing a portfolio of r&d projects. *Naval Research Logistics (NRL)*, *48*(1), 18–40.

Bellman, R. (1954). *The theory of dynamic programming* (Tech. Rep.). DTIC Document.

Beranek, W. (1965). Mathematical programming and the analysis of capital budgeting problems (h. martin weingartner). *SIAM Review*, *7*(2), 306.

Boughaci, D. (2013). Metaheuristic approaches for the winner determination problem in combinatorial auction. In *Artificial intelligence, evolutionary computing and metaheuristics* (pp. 775–791). Springer.

Boughaci, D., Benhamou, B., & Drias, H. (2009). A memetic algorithm for the optimal winner determination problem. *Soft Computing*, *13*(8-9), 905–917.

Boughaci, D., Benhamou, B., & Drias, H. (2010). Local search methods for the optimal winner determination problem in combinatorial auctions. *Journal of Mathematical Modelling and Algorithms*, *9*(2), 165–180.

Boyer, V. (2007). *Contribution à la programmation en nombre entier* (Unpublished doctoral dissertation). INSA de Toulouse.

Boyer, V., Elkihel, M., & El Baz, D. (2009). Heuristics for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, *199*(3), 658–664.

Breu, R., & Burdet, C.-A. (1974). Branch and bound experiments in zero-one programming. In *Approaches to integer programming* (pp. 1–50). Springer.

Caplice, C. G. (1996). *An optimization based bidding process: a new framework for shipper-carrier relationships* (Unpublished doctoral dissertation). Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA.

Caprara, A., & Monaci, M. (2004). On the two-dimensional knapsack problem. *Operations Research Letters*, *32*(1), 5–14.

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d‚Äôéquations simultanées. *Comp. Rend. Sci. Paris*, *25*(1847), 536–538.

Chekuri, C., & Khanna, S. (2005). A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, *35*(3), 713–728.

Chen, L., Chen, X., Hu, S., & Lin, Y. (2008). Hybrid algorithm for winner determination problem in combinatorial auctions. In *Computational intelligence and security, 2008. cis'08. international conference on* (Vol. 2, pp. 82–86).

Chen, S.-H., Chang, P.-C., Cheng, T., & Zhang, Q. (2012). A self-guided genetic algorithm for permutation flowshop scheduling problems. *Computers & Operations Research*, *39*(7), 1450–1457.

Cheng, M. X., Li, Y., & Du, D.-Z. (2006). *Combinatorial optimization in communi-*

cation networks (Vol. 18). Springer Science & Business Media.

Cherbaka, N. S., Meller, R. D., & Ellis, K. P. (2004). Multidimensional knapsack problems and their application to solving manufacturing insourcing problems. In *Iie annual conference. proceedings* (p. 1).

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*, *38*(4), 1338–1350.

Cho, J., & Kim, Y. (1997). A simulated annealing algorithm for resource constrained project scheduling problems. *Journal of the Operational Research Society*, *48*(7), 736–744.

Chu, P. C., & Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, *4*(1), 63–86.

Cotta, C., & Troya, J. M. (1998). A hybrid genetic algorithm for the 0–1 multiple knapsack problem. In *Artificial neural nets and genetic algorithms* (pp. 250–254).

Dammeyer, F., & Voß, S. (1993). Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, *41*(2), 29–46.

Dantzig, G. B. (1957). Discrete-variable extremum problems. *Operations research*, *5*(2), 266–288.

Dantzig, T. (1932). *Number: The language of science (a critical survey written for the cultured non-mathematician)* (G. A. . U. Ltd, Ed.).

Deane, J., & Agarwal, A. (2013). Neural, genetic, and neurogenetic approaches for solving the 0-1 multidimensional knapsack problem. *International Journal of Management & Information Systems (Online)*, *17*(1), 43.

Della Croce, F., & Grosso, A. (2012). Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem. *Computers & Operations Research*, *39*(1), 27–31.

De Vries, S., & Vohra, R. V. (2003). Combinatorial auctions: A survey. *INFORMS Journal on computing*, *15*(3), 284–309.

Djannaty, F., & Doostdar, S. (2008). A hybrid genetic algorithm for the multidimensional knapsack problem. *International Journal of Contemporary Mathematical Sciences*, *3*(9), 443–456.

Dobson, G. (1982). Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research*, *7*(4), 515–531.

Drake, J. H., Özcan, E., & Burke, E. K. (2015). Modified choice function heuristic selection for the multidimensional knapsack problem. In *Genetic and evolutionary computing* (pp. 225–234). Springer.

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

Dueck, G., & Wirsching, J. (1989). *Threshold accepting algorithms for multi-constraint 0-1 knapsack problems*. IBM Deutschland GmbH, Heidelberg Scientific Center.

Dyer, M. E. (1980). Calculating surrogate constraints. *Mathematical Programming*, *19*(1), 255–278.

Escudero, L. F., Landete, M., & Marín, A. (2009). A branch-and-cut algorithm for the winner determination problem. *Decision Support Systems*, *46*(3), 649–659.

Farzi, S. (2010). Discrete quantum-behaved particle swarm optimization for the multi-unit combinatorial auction winner determination problem. *Journal of Applied Sciences*, *10*(4), 291–297.

Fayard, D., & Plateau, G. (1975). Resolution of the 0–1 knapsack problem: comparison of methods. *Mathematical Programming*, *8*(1), 272–307.

Feng, Y., Jia, K., & He, Y. (2014). An improved hybrid encoding cuckoo search algorithm for 0-1 knapsack problems. *Computational intelligence and neuroscience*, *2014*, 1.

Ferreira, C. E., Grötschel, M., Martin, A., Weismantel, R., Kiefl, S., & Krispenz, L. (1993). Some integer programs arising in the design of main frame computers. *Zeitschrift für Operations Research*, *38*(1), 77–100.

Fidanova, S. (2002). Evolutionary algorithm for multidimensional knapsack problem. *Proceedings of PPSNVII*.

Figueira, J. R., Paquete, L., Simões, M., & Vanderpooten, D. (2013). Algorithmic improvements on dynamic programming for the bi-objective {0, 1} knapsack problem. *Computational Optimization and Applications*, *56*(1), 97–111.

Fox, G. E., & Scudder, G. D. (1985). A heuristic with tie breaking for certain 0–1 integer programming models. *Naval Research Logistics Quarterly*, *32*(4), 613–623.

Fréville, A. (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, *155*(1), 1–21.

Fréville, A., & Hanafi, S. (2005). The multidimensional 0-1 knapsack problem,Äîbounds and computational aspects. *Annals of Operations Research*, *139*(1), 195–227.

Freville, A., & Plateau, G. (1986). Heuristics and reduction methods for multiple constraints 0–1 linear programming problems. *European Journal of Operational Research*, *24*(2), 206–215.

Fréville, A., & Plateau, G. (1993). Sac à dos multidimensionnel en variables 0-1: encadrement de la somme des variables à l'optimum. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, *27*(2), 169–187.

Freville, A., & Plateau, G. (1994). An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics*, *49*(1), 189–212.

Fréville, A., & Plateau, G. (1996). The 0-1 bidimensional knapsack problem: Toward an efficient high-level primitive tool. *Journal of Heuristics*, *2*(2), 147–167.

Fujishima, Y., Leyton-Brown, K., & Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches.

In *Ijcai* (Vol. 99, pp. 548–553).

Fukunaga, A. S. (2011). A branch-and-bound algorithm for hard multiple knapsack problems. *Annals of Operations Research*, *184*(1), 97–119.

Gabaldon, E., Lerida, J. L., Guirado, F., & Planes, J. (2014). Slowdown-guided genetic algorithm for job scheduling in federated environments. In *International conference on nature of computation and communication* (pp. 181–190).

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* (pp. 21–30).

Gallo, G., Hammer, P. L., & Simeone, B. (1980). Quadratic knapsack problems. In *Combinatorial optimization* (pp. 132–149). Springer.

Gavish, B., & Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical programming*, *31*(1), 78–105.

Geem, Z. W., Kim, J. H., & Loganathan, G. (2001). A new heuristic optimization algorithm: harmony search. *Simulation*, *76*(2), 60–68.

Gen, M. (2006). Genetic algorithms and their applications. In *Springer handbook of engineering statistics* (pp. 749–773). Springer.

Geoffrion, A. M. (1969). An improved implicit enumeration approach for integer programming. *Operations Research*, *17*(3), 437–454.

Gilmore, P., & Gomory, R. (1966). The theory and computation of knapsack functions. *Operations Research*, *14*(6), 1045–1074.

Glover, F. (1965). A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, *13*(6), 879–919.

Glover, F. (1968). Surrogate constraints. *Operations Research*, *16*(4), 741–749.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, *13*(5), 533–549.

Glover, F. (1989). Tabu search-part i. *ORSA Journal on computing*, *1*(3), 190–206.

Glover, F. (1995). *Tabu search fundamentals and uses*. Graduate School of Business, University of Colorado Boulder.

Glover, F., & Kochenberger, G. A. (1996). Critical event tabu search for multidimensional knapsack problems. In *Meta-heuristics* (pp. 407–427). Springer.

Glover, F., & Laguna, M. (2013). *Tabu search*. Springer.

Gonen, R., & Lehmann, D. (2002). Linear programming helps solving large multi-unit combinatorial auctions. *arXiv preprint cs/0202016*.

Gottlieb, J. (1999). On the effectivity of evolutionary algorithms for the multidimensional knapsack problem. In *European conference on artificial evolution* (pp. 23–37).

Graves, R. L., Schrage, L., & Sankaran, J. (1993). An auction method for course registration. *Interfaces*, *23*(5), 81–92.

Greenberg, H. J., & Pierskalla, W. P. (1970). Surrogate mathematical programming. *Operations Research*, *18*(5), 924–939.

Guo, Y., Lim, A., Rodrigues, B., & Tang, J. (2006). Using a lagrangian heuristic for a combinatorial auction problem. *International Journal on Artificial Intelligence Tools*, *15*(03), 481–489.

Guo, Y., Lim, A., Rodrigues, B., & Zhu, Y. (2004). Heuristics for a brokering set packing problem.

Guo, Y., Lim, A., Rodrigues, B., & Zhu, Y. (2006). Heuristics for a bidding problem. *Computers & operations research*, *33*(8), 2179–2188.

Gupta, C. (2008). *Optimization techniques in operation research*. IK International Pvt Ltd.

Hanafi, S., & Freville, A. (1998). An efficient tabu search approach for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, *106*(2), 659–675.

Hanafi, S., & Wilbaut, C. (2008). Scatter search for the 0–1 multidimensional knapsack problem. *Journal of Mathematical Modelling and Algorithms*, *7*(2), 143–159.

Hansen, P., & Mladenović, N. (1999). An introduction to variable neighborhood search. In *Meta-heuristics* (pp. 433–458). Springer.

Hartmanis, J. (1982). Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, *24*(1), 90.

Hembecker, F., Lopes, H. S., & Godoy Jr, W. (2007). Particle swarm optimization for the multidimensional knapsack problem. In *International conference on adaptive and natural computing algorithms* (pp. 358–365).

Hill, R. R., Cho, Y. K., & Moore, J. T. (2012). Problem reduction heuristic for the 0–1 multidimensional knapsack problem. *Computers & Operations Research*, *39*(1), 19–26.

Hoff, A., Løkketangen, A., & Mittet, I. (1996). Genetic algorithms for 0/1 multidimensional knapsack problems. In *Proceedings norsk informatikk konferanse* (pp. 291–301).

Holland, A., & O'sullivan, B. (2004). Towards fast vickrey pricing using constraint programming. *Artificial Intelligence Review*, *21*(3-4), 335–352.

Holte, R. C. (2001). Combinatorial auctions, knapsack problems, and hill-climbing search. In *Conference of the canadian society for computational studies of intelligence* (pp. 57–66).

Hooker, J. N., Yan, H., Grossmann, I. E., & Raman, R. (1994). Logic cuts for processing networks with fixed charges. *Computers & operations research*, *21*(3), 265–279.

Hoos, H., & Stutzle, T. (n.d.). *Stochastic local search: Foundations and applications (2005).* Elsevier/Morgan Kaufmann, San Francisco. Mathematical Methods Of Operation Research, ISBN.

Hoos, H. H., & Boutilier, C. (2000). Solving combinatorial auctions using stochastic local search. In *Aaai/iaai* (pp. 22–29).

Horowitz, E., & Sahni, S. (1974). Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, *21*(2), 277–292.

Huston, S., Puchinger, J., & Stuckey, P. (2008). The core concept for 0/1 integer programming. In *Proceedings of the fourteenth symposium on computing: the australasian theory-volume 77* (pp. 39–47).

Jackson, C. (1976). *Technology for spectrum markets* (Unpublished doctoral dissertation). Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA.

Jat, S. N., & Yang, S. (2009). A guided search genetic algorithm for the university course timetabling problem.

Johnson, E. L., Kostreva, M. M., & Suhl, U. H. (1985). Solving 0-1 integer programming problems arising from large scale planning models. *Operations Research*, *33*(4), 803–819.

Jones, J. (2000). *Incompletely specified combinatorial auction: an alternative allocation mechanism for business-to-business negotiations* (Unpublished doctoral dissertation). Warrington College of Business Administration, University of Florida,, Gainesville, FL, USA.

Kameshwaran, S., & Benyoucef, L. (2006). Branch on price: A fast winner determination algorithm for discount auctions. In *International conference on algorithmic applications in management* (pp. 375–386).

Karwan, M. H., & Rardin, R. L. (1979). Some relationships between lagrangian and surrogate duality in integer programming. *Mathematical Programming*, *17*(1), 320–334.

Ke, L., Feng, Z., Ren, Z., & Wei, X. (2010). An ant colony optimization approach for the multidimensional knapsack problem. *Journal of Heuristics*, *16*(1), 65–83.

Kellerer, H., Pferschy, U., & Pisinger, D. (n.d.). *Knapsack problems. 2004.* Springer, Berlin.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004a). *Introduction to np-completeness of knapsack problems*. Springer.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004b). Other knapsack problems. In *Knapsack problems* (pp. 389–424). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-540-24777-7_13

Kelly, T. (2004). Generalized knapsack solvers for multi-unit combinatorial auctions: Analysis and application to computational resource allocation. In *International workshop on agent-mediated electronic commerce* (pp. 73–86).

Khuri, S., Bäck, T., & Heitkötter, J. (1994). The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the 1994 acm symposium on applied computing* (pp. 188–193).

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983). Optimization by simmulated annealing. *science*, *220*(4598), 671–680.

Kocay, W., & Kreher, D. L. (2004). *Graphs, algorithms, and optimization*. CRC Press.

Kochenberger, G. A., McCarl, B. A., & Paul Wyman, F. (1974). A heuristic for general integer programming. *Decision Sciences*, *5*(1), 36–44.

Kolesar, P. J. (1967). A branch and bound algorithm for the knapsack problem. *Management science*, *13*(9), 723–735.

Kong, M., & Tian, P. (2006). Apply the particle swarm optimization to the multidimensional knapsack problem. In *International conference on artificial intelligence and soft computing* (pp. 1140–1149).

Kong, M., Tian, P., & Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional knapsack problem. *Computers & Operations Research*, *35*(8), 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*, *63*, 7–22.

Layeb, A. (2013). A hybrid quantum inspired harmony search algorithm for 0–1 optimization problems. *Journal of Computational and Applied Mathematics*, *253*, 14–25.

Leguizamon, G., & Michalewicz, Z. (1999). A new version of ant system for subset problems. In *Evolutionary computation, 1999. cec 99. proceedings of the 1999 congress on* (Vol. 2).

Lemke, C. E., & Spielberg, K. (1967). Direct search algorithms for zero-one and mixed-integer programming. *Operations Research*, *15*(5), 892–914.

Leyton-Brown, K., Shoham, Y., & Tennenholtz, M. (2000). An algorithm for multi-unit combinatorial auctions. In *Aaai/iaai* (pp. 56–61).

Li, H., Jiao, Y.-C., Zhang, L., & Gu, Z.-W. (2006). Genetic algorithm based on the orthogonal design for multidimensional knapsack problems. In *International conference on natural computation* (pp. 696–705).

Løkketangen, A., Jörnsten, K., & Storøy, S. (1994). Tabu search within a pivot and complement framework. *International transactions in operational research*, *1*(3), 305–316.

Lorie, J. H., & Savage, L. J. (1955). Three problems in rationing capital. *The journal of business*, *28*(4), 229–239.

Louis, S., & Li, G. (1997). Augmenting genetic algorithms with memory to solve traveling salesman problems. In *Proceedings of the joint conference on information sciences* (pp. 108–111).

Lust, T., & Teghem, J. (2012). The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, *19*(4), 495–520.

Magazine, M., & Oguz, O. (1984). A heuristic algorithm for the multidimensional zero-one knapsack problem. *European Journal of Operational Research*, *16*(3), 319–326.

Mansini, R., & Speranza, M. (2002). A multidimensional knapsack model for asset-backed securitization. *Journal of the Operational Research Society*, *53*(8), 822–832.

Markowitz, H. M., & Manne, A. S. (1957). On the solution of discrete programming problems. *Econometrica: journal of the Econometric Society*, 84–110.

Marsten, R. E., & Morin, T. L. (1977). Optimal solutions found for senju and toyoda's 0/1 integer programming problems. *Management Science*, 1364–1365.

Marsten, R. E., & Morin, T. L. (1978). A hybrid approach to discrete mathematical programming. *Mathematical programming*, *14*(1), 21–40.

Martello, S., & Toth, P. (1977). An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, *1*(3), 169–175.

Martello, S., & Toth, P. (1979). The 0-1 knapsack problem. In N. Christofides, A. Mingozzi, P. Toth, & C. Sandi (Eds.), *Combinatorial optimization* (p. 237-279). John Wiley & Sons.

Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc.

Mathews, G. B. (1896). On the partition of numbers. In *Proceedings of the london mathematical society 28* (pp. 486–490).

Mcmillan, C., & Plane, D. R. (1973). Resource allocation via 0,Äì1 programming. *Decision Sciences*, *4*(1), 119,Äì132. doi: 10.1111/j.1540-5915.1973.tb01710.x

Meaning of combinatorial optimization. (2007). MIT - Press Statistics Glossary. Retrieved from http://www.encyclo.co.uk/meaning-of-combinatorialoptimization (consulted 01/07/2016)

Meier, H., Christofides, N., & Salkin, G. (2001). Capital budgeting under uncertainty-an integrated approach using contingent claims analysis and integer programming. *Operations Research*, *49*(2), 196–206.

Nakbi, W., Alaya, I., & Zouari, W. (2015). A hybrid lagrangian search ant colony optimization algorithm for the multidimensional knapsack problem. *Procedia Computer Science*, *60*, 1109–1119.

Nauss, R. M. (1976). An efficient algorithm for the 0-1 knapsack problem. *Management Science*, *23*(1), 27–31.

Nisan, N. (2000). A partitioned stochastic search algorithm: application to multi-unit winner determination problem in combinatorial auction. In *Proceedings of the 2nd acm conference on electronic commerce* (pp. 1–12).

Osorio, M. A., Glover, F., & Hammer, P. (2002). Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Annals of Operations Research*, *117*(1-4), 71–93.

Pan, Q.-K., Suganthan, P. N., Tasgetiren, M. F., & Liang, J. J. (2010). A self-adaptive global best harmony search algorithm for continuous optimization problems. *Applied Mathematics and Computation*, *216*(3), 830–848.

Pappa, G. L., & Freitas, A. A. (2009). *Automating the design of data mining algorithms: an evolutionary computation approach*. Springer Science & Business Media.

Petersen, C. C. (1967). Computational experience with variants of the balas algorithm applied to the selection of r&d projects. *Management Science*, *13*(9), 736–750.

Pirkul, H. (1987). A heuristic solution procedure for the multiconstraint zero? one knapsack problem. *Naval Research Logistics*, *34*(2), 161–172.

Pisinger, D. (1995a). An expanding-core algorithm for the exact 0–1 knapsack problem. *European Journal of Operational Research*, *87*(1), 175–187.

Pisinger, D. (1995b). A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, *83*(2), 394–410.

Pisinger, D. (1997). A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, *45*(5), 758–767.

Pisinger, D. (1999). Core problems in knapsack algorithms. *Operations Research*, *47*(4), 570–575.

Pisinger, D. (2000). A minimal algorithm for the bounded knapsack problem. *INFORMS Journal on Computing*, *12*(1), 75–82.

Pisinger, D. (2005). Where are the hard knapsack problems? *Computers & Operations Research*, *32*(9), 2271–2284.

Plateau, G., & Elkihel, M. (1985). A hybrid method for the 0-1 knapsack problem.

Methods of Operations Research, *49*, 277–293.

Poirriez, V., Yanev, N., & Andonov, R. (2009). A hybrid algorithm for the unbounded knapsack problem. *Discrete Optimization*, *6*(1), 110–124.

Puchinger, J., & Raidl, G. R. (2008). Bringing order into the neighborhoods: relaxation guided variable neighborhood search. *Journal of Heuristics*, *14*(5), 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* (pp. 195–208).

Puchinger, J., Raidl, G. R., & Pferschy, U. (2010). The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, *22*(2), 250–265.

Qian, F., & Ding, R. (2007). Simulated annealing for the 0/1 multidimensional knapsack problem. *NUMERICAL MATHEMATICS-ENGLISH SERIES-*, *16*(4), 320.

Raidl, G. R., & Gottlieb, J. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, *13*(4), 441–475.

Rao, S. S., & Rao, S. (2009). *Engineering optimization: theory and practice*. John Wiley & Sons.

Rasheed, K. (1999). Guided crossover: A new operator for genetic algorithm based optimization. In *Evolutionary computation, 1999. cec 99. proceedings of the 1999 congress on* (Vol. 2).

Rassenti, S. J., Smith, V. L., & Bulfin, R. L. (1982). A combinatorial auction mechanism for airport time slot allocation. *The Bell Journal of Economics*, 402–417.

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* (pp. 298–304).

Rong, A., & Figueira, J. R. (2012). Computational performance of basic state reduction based dynamic programming algorithms for bi-objective 0–1 knapsack problems. *Computers & Mathematics with Applications*, *63*(10), 1462–1480.

Rothkopf, M. H., Pekeč, A., & Harstad, R. M. (1998). Computationally manageable combinational auctions. *Management science*, *44*(8), 1131–1147.

Rudolph, G., & Sprave, J. (1996). Significance of locality and selection pressure in the grand deluge evolutionary algorithm. In *International conference on parallel problem solving from nature* (pp. 686–695).

Rustagi, J. S. (2014). *Optimization techniques in statistics*. Elsevier.

Sandholm, T., & Suri, S. (2003). Bob: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, *145*(1), 33–58.

Sandholm, T., Suri, S., Gilpin, A., & Levine, D. (2002). Winner determination in combinatorial auction generalizations. In *Proceedings of the first international joint conference on autonomous agents and multiagent systems: part 1* (pp. 69–76).

Sandholm, T., Suri, S., Gilpin, A., & Levine, D. (2005). Cabob: A fast optimal algorithm for winner determination in combinatorial auctions. *Management Science*, *51*(3), 374–390.

Sarin, S., Karwan, M. H., & Rardin, R. L. (1987). A new surrogate dual multiplier search procedure. *Naval Research Logistics (NRL)*, *34*(3), 431–450.

Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, *12*, 1–68.

Senju, S., & Toyoda, Y. (1968). An approach to linear programming with 0-1 variables. *Management Science*, B196–B207.

Sheffi, Y. (2004). Combinatorial auctions in the procurement of transportation services. *Interfaces*, *34*(4), 245–252.

Shih, W. (1979). A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, *30*(4), 369–378.

Singh, R., Sen, A., & Sarkar, U. (2012). Bidding and allocation in combinatorial auctions. In *Proceedings of the athens: Atiner‚Äôs conference.*

Sinha, P., & Zoltners, A. A. (1979). The multiple-choice knapsack problem. *Operations Research*, *27*(3), 503–515.

Snášel, V., Dvorskỳ, J., Ochodková, E., Krömer, P., Platoš, J., & Abraham, A. (2010). Evolving quasigroups by genetic algorithms. In *proceedings of the dateso 2010 workshop* (pp. 108–117).

Snyman, J. (2005). *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms* (Vol. 97). Springer Science & Business Media.

Sural, H., Van Wassenhove, L. N., Potts, C. N., et al. (1997). *The bounded knapsack problem with setups*. INSEAD.

Thesen, A. (1975). A recursive branch and bound algorithm for the multidimensional knapsack problem. *Naval Research Logistics Quarterly*, *22*(2), 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*, *32*(4), 226–242.

Toth, P. (1980). Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, *25*(1), 29–45.

Toyoda, Y. (1975). A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, *21*(12), 1417–1427.

Trauth Jr, C., & Woolsey, R. (1969). Integer linear programming: a study in computational efficiency. *Management Science*, *15*(9), 481–493.

Tuo, S., Yong, L., & Deng, F. (2014). A novel harmony search algorithm based on teaching-learning strategies for 0-1 knapsack problems. *The Scientific World Journal*, *2014*.

Ulungu, E. L., & Teghem, J. (1994). Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, *3*(2), 83–

104.

Varnamkhasti, M. J. (2012). Overview of the algorithms for solving the multidimensional knapsack problems. *Advanced Studies in Biology*, *4*(1), 37–47.

Vasquez, M., & Hao, J.-K. (2001). Une approche hybride pour le sac à dos multidimensionnel en variables 0–1. *RAIRO-Operations Research*, *35*(4), 415–438.

Vasquez, M., Hao, J.-K., et al. (2001). A hybrid approach for the 0-1 multidimensional knapsack problem. In *Ijcai* (pp. 328–333).

Vasquez, M., & Vimont, Y. (2005). Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, *165*(1), 70–81.

Vázquez-Barreiros, B., Mucientes, M., & Lama, M. (2014). A genetic algorithm for process discovery guided by completeness, precision and simplicity. In *International conference on business process management* (pp. 118–133).

Veni, K. K., & Balachandar, S. R. (2010). A new heuristic approach for large size zero–one multi knapsack problem using intercept matrix. *Int. J. Comput. Math. Sci*, *4*(5), 259–263.

Volgenant, A., & Zoon, J. (1990). An improved heuristic for multidimensional 0-1 knapsack problems. *Journal of the Operational Research Society*, *41*(10), 963–970.

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*, *48*, 17–23.

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

Weingartner, H. M., & Ness, D. N. (1967). Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, *15*(1), 83–103.

Xiang, W.-l., An, M.-q., Li, Y.-z., He, R.-c., & Zhang, J.-f. (2014). A novel discrete global-best harmony search algorithm for solving 0-1 knapsack problems. *Discrete Dynamics in Nature and Society*, *2014*.

Yang, H., Wang, M., & Yang, C. (2013). A hybrid of rough sets and genetic algorithms for solving the 0-1 multidimensional knapsack problem. *Innovative Computing Information and Control*, *9*(9), 3537–3548.

Yang, S., & Jat, S. N. (2011). Genetic algorithms with guided and local search strategies for university course timetabling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *41*(1), 93–106.

Yoon, Y., & Kim, Y.-H. (2013). A memetic lagrangian heuristic for the 0-1 multidimensional knapsack problem. *Discrete Dynamics in Nature and Society*, *2013*.

Yoon, Y., Kim, Y.-H., & Moon, B.-R. (2005). An evolutionary lagrangian method for the 0/1 multiple knapsack problem. In *Proceedings of the 7th annual conference on genetic and evolutionary computation* (pp. 629–635).

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*, *218*(2), 366–376.

Yu, G. (2013). *Industrial applications of combinatorial optimization* (Vol. 16). Springer Science & Business Media.

Zhang, B., Pan, Q.-K., Zhang, X.-L., & Duan, P.-Y. (2015). An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems. *Applied Soft Computing*, *29*, 288–297.

Zhang, Q. (2009). A guided memetic algorithm with probabilistic models. *International Journal of Innovative Computing, Information and Control*, *5*(12 (B)), 4753–4764.

Zhang, Q., Sun, J., & Tsang, E. (2005). An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE transactions on evolutionary computation*, *9*(2), 192–200.

Zheng, G., & Lin, Z. (2012). A winner determination algorithm for combinatorial auctions based on hybrid artificial fish swarm algorithm. *Physics Procedia*, *25*, 1666–1670.