Cooperative Approaches between some Metaheuristics and Integer programming for solving Generalized Multiple Knapsack Problem with Setup and its variants

Yassine Adouani

Republic of Tunisia
Ministry of Higher Education
and Scientific Research

University of Sfax
Faculty of Economics and
Management of Sfax

PhD THESIS

Submitted in partial fulfilment of the requirements for the degree of

DOCTOR IN

Management sciences

Specialty: Operational research and decision making

Cooperative Approaches between some Metaheuristics and Integer programming for solving Generalized Multiple Knapsack Problem with Setup and its variants

Presented and publicly defended on 29 July 2020 by:

Yassine ADOUANI

Jury Member

Mr. Nejib HACHICHA	Full Professor, FSEG-Sfax	Chair
Mr. Bassem JARBOUI	Full Professor, IHEC-Sfax	Supervisor
Mr. Abdelkarim EllOUMI	Associate Professor, FSEG-Sfax	Reviewer
Mr. Souhail DHOUIB	Full Professor, ISGI-Sfax	Reviewer
Mr. Abdelaziz DAMMEK	Full Professor, FSEG-Sfax	Member
Mr. Malek MASMOUDI	Associate Professor JMSE-France	Invited Member

Academic year: **2019-2020**

Acknowledgement

It is with great pleasure that i reserve this page as a sign of deep gratitude to all those who have kindly provided the necessary support for the smooth running of this thesis.

I present my thanks to Prof. *Nejib hachicha* for the honor he had accorded me for agreeing to be the committee chair of my thesis. I also thank Prof. *Abdelaziz dammak* for the valuable service to examine my thesis and to be a member of the committee. My distinguished thanks go also to Prof. *Abdelkarim Elloumi* and Prof. *Souhail Dhouib* for taking their time to review my dissertation and for their relevant comments.

I would like to express my deep gratitude to my supervisor Prof. *Bassem Jarboui* for his outstanding commitment to this thesis. I am also grateful for the support he gave me. His professionalism, friendliness and pedagogical and scientific qualities have been invaluable.

I have the favor to thank my supervisor Prof. *Malek Masmoudi* for his interesting advices which are very useful to me and his collaboration to well accomplish this work. I am thankful to all my colleagues at MODILS Laboratory (Sfax-Tunisia).

My success would not have been possible without the love, patience, prayers and support of my parents Messaoud and Mabrouka. I would like also to thank my sister Malika, my brothers Soufien, Nebil, Khaled and Tarek.

Finally, I would like to express my deepest and heartfelt thanks to my beloved wife, Sana Hamdi, and my son, Nader.

AUTHOR'S PUBLICATIONS

The contributions proposed in this thesis have been presented in scientific communications and articles.

Publications in international peer-reviewed journals

- Adouani Y., Jarboui B., Masmoudi M. (2019). An efficient new matheuristic to solve the generalized multiple Knapsack Problem with Setup. European journal of industrial Engineering.Vol , pp. 1-27. (IF, 1.26).
- Adouani Y., Jarboui B., Masmoudi M. (2019). A matheuristic to solve the 0-1 generalized quadratic multiple Knapsack Problem with Setup. *Optim Lett*. Vol, pp.1-22. (IF, 1.5).
- Adouani Y., Jarboui B., Masmoudi M. Iterated local search-based matheuristic for the Multiple choice knapsack problem with setup. *Submitted* in International Transactions in Operational Research.
- Adouani Y., Jarboui B., Masmoudi M. Estimation Distribution Algorithm-based Matheuristic for the Multiple Knapsack Problem with Setup. *Submitted* in European journal operation research.

Book Chapter

- Adouani Y., Jarboui B., Masmoudi M. (2019) A Variable Neighborhood Search with Integer Programming for the Zero-One Multiple-Choice Knapsack Problem with Setup. In: Sifaleras A., Salhi S., Brimberg J. (eds) Variable Neighborhood Search. ICVNS 2018. Lecture Notes in Computer Science. Vol. 11328, pp. 152-164. Springer, Cham. (SJR, 0.28).

Publications in international peer-reviewed conferences

- Y. Adouani, B. Jarboui, M. Masmoudi, A Variable neighborhood search with integer programming for the zero-one Multiple-Choice Knapsack Problem with Setup, in the conference 6th International Conference on Variable Neighborhood Search (ICVNS 2018), Sithonia, Halkidiki, Greece, October 4-7, 2018.

- Y. Adouani, M. Masmoudi, I. Alghoul and B. Jardoui, A hybrid approach for zero-one Multiple -Choice Knapsack Problem with Setup, the International Conference of the African Federation of Operational Research Societies, 2-4 july 2018, Tunis.

TABLE OF CONTENTS

Introduction

Problems and motivation

Combinatorial optimization problems allow to model and solve a variety of real life situations. For example, finding a route minimizing the distance can be modeled by a problem of this class. Nevertheless, considering only one objective to optimize may not be sufficient to represent the complexity of real life situations. Indeed, if a company is interested in maximizing its profit, it may also be interested in minimizing its ecological impact. Then several objectives have to be considered. If no preference is given a priori, all solutions such that it is not possible to improve an objective without degrading another one should be returned to the decision maker. After the solving process, the decision maker chooses among the returned solutions.

Many practical situations can be modeled as combinatorial optimization problems. Among these problems, we can find some problems belonging to the knapsack family. The 0-1 Knapsack Problem (KP) is one of the paradigmatic problems in combinatorial optimization where a set of items with given profits and weights is available and the aim is to select a subset of the items in order to maximize the total profit without exceeding a known knapsack capacity. Martello and Toth [77] provide extensive reviews of the major classes of KPs. The 0-1 Knapsack Problem with Setups (KPS) originally introduced in [20] can be seen as a generalization of KP where items belong to disjoint classes and can be selected only if the corresponding class is activated. The selection of a class involves setup costs and resource consumptions thus affecting both the objective function and the capacity constraint. KPS has many applications of interest such as make-to-order production contexts, cargo loading and product category management among others and more generally for allocation resources

problems involving classes of elements [21]. Another application of KPS is originated within the smart-home paradigm where the goal of an efficient management of the buildings energy consumptions is a strong component (see Project FLEXMETER from: http://exmeter.polito.it).

The Multiple Knapsack Problem with Setup (MKPS) can be considered as a set of knapsack problems with different capacities in which a set of disjoint classes of items with knapsack-dependent profits and given weights are available. An item can be selected only if the corresponding class is activated and a class can only be set up in one knapsack. A key feature is that the activation of a class incurs a knapsack-dependent setup cost that should be considered both in the objective function and constraints. The setup cost varies with the knapsack. A solution to the MKPS consists in selecting appropriate items, from different disjoint classes, to enter a knapsack while maximizing its value and respecting its capacity.

Like most knapsack problems, the MKPS finds its application in several concrete industrial problems, e.g., production planning [104], aviation security system [79], etc. For instance, consider a supplier of hollow glass in the agro-alimentary glass packing industry, producing several types of products, including bottles, flacons, and pots [21]. The most important phase in the manufacturing process is the shaping. Indeed, to change the production from one product class to another, the production machinery must be set up and moulds must be changed in the moulding machine. There is no setup between products in the same class. These changes in the manufacturing process require significant setup time and costs. Accordingly, the company needs to decide on how to choose orders so as to maximize the total profit. This represents a typical case involving a Knapsack Problem with Setup (KPS). However, if orders can be served in different periods, but a product class can only be produced in a single period, the cost would depend on the completion time of the order. There would be an initial cost for an order delivered on the client desired date and penalties for delay or precociousness for postponed delivery dates. These costs would depend on the modification of the desired date. Because of the cost variability dependent on the production planning, this problem is more complex than the KPS. Indeed, before denying a production schedule, and in order to maximize its total profit, the company should take into consideration the production capacity, the profit of different products, and the cost of each class at each period. In this case, the problem can be modeled as an MKPS. The KPS is a reduction of the

MKPS when only one production period is considered. Another application of the MKPS arises in the cloud computing industry that faces several decision-making issues that need to be optimized. Hence, the extension of MKPS when a product class can be produced in a multiple periods is a real case study of GMKPS. Prices varies according to the customers expectation of products delivery date i.e. some customers are willing to pay a higher price for a short lead-time while others are willing to wait for their products in exchange for lower prices. Thus, price, delivery period and total profit have very complex connections that are of extreme interest to businesses today. Thus, we consider that orders could be realized in multiple periods, and the products' price depends on the orders' completion time i.e. penalties are added to the initial price in case where products are not delivered at customers' desired due date. In addition, the products (items) could be classified into classes regarding specially their setups i.e. setup is null between products from the same class. The profit for order *j* of class *i* processed in period *t* is and varies for different periods, but the processing time stays the same. To find the assignment of orders that maximizes the total profit, we have to consider the marginal profit of each job, the current production capacity per period, and the setup cost and time from orders. This realistic production scheduling problem is typically our GMKPS case study.

The motivation of this thesis is to introduce a new variant of the knapsack problem with setup (KPS). We refer to it as the generalized multiple knapsack problem with setup (GMKPS) and develop new matheuristics methods combining variable neighborhood search with integer programming to solve the linear problem GMKPS and its variants such as: linear problems MKPS and MCKS and quadratic variant GQMKP. Because of the difficulty of these problems, we are searching for approximate solution techniques with fast solution times for its large scale instances. A promising way to solve the GMKPS, MKPS, MCKS and GQMKP is to consider some techniques based upon the principle of cooperative approach can be viewed as matheuristic that combining neighborhood search techniques with integer programming (IP). Although such techniques produce approximate solution methods, they allow us to present fast algorithms that yield interesting solutions within a short average running time, that is, to generate approximations of good quality to the efficient set. We will see in an overview about the methods for solving knapsack problems family that many metaheuristics have already been adapted to tackle MKPS problems. But most of the methods

include many parameters and are sometimes so complex that it is difficult to deeply understand the behavior of these methods. It makes the application of these methods to MKPS problems hard and not necessary efficient. For the new methods developed in this thesis, two features are expected: simplicity and effectiveness. The methods should be as simple as possible to easily adapt them to different MKPS problems and to give better results as state-of-the-art results on different MKPS problems. We also intend to give through this work a better knowledge concerning the efficient solutions of MKPS problems, as well as introducing new techniques to solve new MKPS problems. Another motivation is to apply the methods developed to real MKPS problems.

Solution overview and contributions

Many solution methods have been designed for the KP and its variants: (i) solving the given problem using exact methods and/or (ii) searching near optimal solutions using metaheuristic methods. An exact algorithm tries to find an optimal or a set of optimal solutions for a given problem. For the problems belonging to the knapsack family, an optimal solution can be found using branch and bound, branch and cut, and/or dynamic programming methods. Nevertheless, for large-scale problems, an exact method might need exponential computation time. This often leads to a solution time that is too high for the practical situation. Thus, the development of metaheuristic methods has received more attention in the last decades, however, comes at the price of having no guarantee about their quality. For that reason, we define new approaches that combine exact and metaheuristic methods. These methods, noted as cooperative approaches, represent a powerful tool for solving combinatorial optimization problems. The GMKPS, MKPS, MCKS and GQMKP are NP-hard combinatorial problems since it is a generalization of the standard 0-1 KP, which is known to be an NP-hard problem [68, 77] exact methods would be rather inefficient in solving large-size instances of the four problems cited above. An alternative to exact methods would be to combine exact and metaheuristic algorithms. This cooperative approach, referred to as *matheuristics*, seems to be a very promising path towards the solution of rich combinatorial optimization problems. Matheuristics take advantage from synergy between approximate and exact solution approaches and often lead to considerably higher performance with respect to

solution quality and running time. However, adapting those mechanisms to different problems can be challenging. In this thesis, we will propose to design and implement a matheuristic framework to solve GMKPS, MKPS and MCKS, and show how it can be improved to solve related rich quadratic variant GQMKP.

The main objective of this thesis is to provide a solving approaches for the GMKPS and its variants. We introduce a mixed Integer programming (MIP) formulation that, due to the complexity of the GMKPS, cannot solve even small test instances. In fact, it is usually difficult to assign items to the whole sets of knapsacks. In addition, the consideration of the knapsack-dependent cost related to each class of products and the knapsack-dependent profit associated to each item increases the complexity of the problem. Therefore, the design of a new approach providing high quality solutions in a reasonable computing time is quite challenging. An alternative to exact methods would be to develop a first cooperative approach, can be viewed as *matheuristic* that combine a variable neighborhood descent (VND) with an exact solving technique: local search techniques to include classes to knapsacks and integer programming (IP) to include items in each knapsack. Experimental results show the efficiency and the performance of the proposed approach on randomly generated instances of GMKPS. Furthermore, we enhance our solution approach combining local search techniques with integer programming. We carry out a computational study to assess the performance of the proposed cooperative approach on a new set of instances from MKPS. The challenge of the second cooperative approach is to propose an efficient cooperative framework between variable neighborhood search VNS and Integer programming to solve the linear problem MCKS. Finally, the third cooperative approach addressed to solve the quadratic variant GQMKP. The attempt of the third cooperative is to combine new efficient *Matheuristic* VNS and integer programming. The computational results shows that the proposed cooperative approaches (or matheuristics) are competitive compared with the state-of-the-art methods. The different contributions are listed below:

1) We introduce a new variant of the knapsack problem with setup (KPS). We refer to it as the generalized multiple knapsack problems with setup (GMKPS). GMKPS originates from industrial production problems where the items are divided into classes and processed in multiple periods. We refer to the particular case, where items from the same class cannot be processed in more than one period, as the multiple

knapsack problems with setup (MKPS). First, we provide mathematical formulations of GMKPS and MKPS and provide an upper bound expression for the knapsack problem. We then propose a cooperative approach (matheuristic) that combines variable neighborhood descent (VND) with integer programming (IP). We consider local search techniques to assign classes to knapsacks and apply the IP to select the items in each knapsack. Computational experiments on randomly generated instances show the efficiency of our matheuristic in comparison to the direct use of a commercial solver.

2) The challenge of the second cooperative approach is to develop an algorithm combining VNS with IP to solve MCKS. The idea consists in partitioning a MCKS solution into two levels. The first level contains the classes (or setup variables) to be fixed by the VNS, where the second level contains the remainder of variables (items) that will be optimally optimized by the Integer programming. For the numerical experiment, we generated different instances for MCKS. In the experimental setting, we compared our cooperative approach to the Mixed Integer Programming provided in literature. Experimental results clearly showed the efficiency and effectiveness of our approach.
3) We use a linearization technique of the existing mathematical model and we propose a new cooperative approach combining matheuristic variable neighborhood search (MVNS) with integer programing (IP) to solve the generalized quadratic multiple knapsack problem (GQMKP). The matheuristic considers a local search technique with an adaptive perturbation mechanism based on a mathematical programming to assign the classes to different knapsacks, and then once the assignment is identified, applies the IP to select the items to allocate to each knapsack. Experimental results obtained on a wide set of benchmark instances clearly show the competitiveness of the proposed approach compared to the best state-of-the-art solving techniques.

Thesis structure

The thesis contains four main parts. The first part presents an overview of the main cooperative approaches. The second part is dedicated to the development of a new cooperative approach between variable neighborhood descent (VND) and Integer programming (IP), to solve the (G)MKPS. The third provides a new efficient cooperative approach between variable neighborhood search (VNS) and IP to solve MCKS. The fourth part discusses a new hybrid approach in which mathematical programming is an embedded component into a variable neighborhood search (MVNS) that has the ability to solve the quadratic variant of GMKPS, denoted by GQMKP.

More specifically, the thesis is organized as follows. A bibliographic study which aims to present an overview of the exact methods, metaheuristic and cooperative approaches and explain their adaptation for evolving programs is provided in first part (Chapter I). Section I.2 discusses the exact methods, while section I.3 presents the (meta-)heuristics approaches used to solve the knapsack problems family. Finally, section I.4 and I.5 provide an overview of the cooperative and matheuristic approaches. We give a general presentation of the integer programming and local search techniques forming the core of our solutions approaches, with section I.6 concluding. The remaining chapters describe the methodological contributions of this thesis. Chapter II is about the GMKPS. We formally introduce the problem. Then, we propose a mixed integer linear programming formulation and an integer model based on the Dantzig-Wolfe decomposition. In Section II.2, the related literature of the problem is presented. Section II.3 contains the mathematical formulations of GMKPS and their particular case MKPS. In Section II.4, we propose a cooperative approach can be seen as *matheuristic* that combine variable neighborhood descent (VND) and integer programming (IP) for the (G)MKPS. The experimental results and their interpretations are reported in Section II.5. In Section II. 6, we conclude the chapter and give possible and future research ideas. In Chapter III we move from the MCKS problem and apply matheuristic (or cooperative) approach combining VNS with IP to solve this problem. In Section III.1, the presentation and related literature of the problem are presented. Section III.2 contains the mathematical formulations of MCKS. In Section III.3, we propose a matheuristic approach combining VNS and integer programming for MCKS. The experimental results and their interpretations are reported in Section III.4. In Section III.5, we conclude the chapter and give possible and future research

ideas. Chapter IV is devoted to the description of cooperative solution approach to solve the GQMKP. We analyze the challenges encountered while developing the cooperative approaches between some Local Search techniques and Integer programming and provide a simple and effective data structure which may be easily generalized for quadratic variant of GMKPS problem. Later, we improved the efficiency of the proposed approaches: VND&IP and VNS&IP on a set of new generated instances for (G)MKPS and MCKPS. We provide a sensitivity analysis distinguishing the main components for increasing the performance of our cooperative approaches. In Section VI.1, the presentation and related literature of the problem are presented. Section IV.2 contains the mathematical formulation of the GQMKP. Section IV.3 contains our cooperative approach combining MVNS with IP. The experimental results and their interpretations are reported in Section IV.4 and, finally, the conclusions are outlined in Section IV.5. Finally, overall conclusions and perspectives are drawn in the last chapter of the thesis. In Appendix A, we report detailed computational experiments carried out in this thesis.

Chapter I

Cooperative approaches

I.1 Introduction

This chapter provides an overview of different methods for solving combinatorial optimization problems [30]. It is not so easy to classify the existing optimization methods. Beyond the classical separation between exact methods and (meta-) heuristic methods, several papers are devoted to the taxonomy of cooperative approach. Cooperative (or Hybrid) methods are not new in the operational research community. This class of cooperative approaches includes several sub classes among which techniques combining (meta-) heuristics and exact algorithms have a dominating place.

In the remainder of this chapter, we elaborate further on exacts method and (meta-) heuristics approaches and explain some differences among different techniques and paradigms. We then focus on the context of cooperative approach, the paradigm, general framework, steps, and different components.

I.2 Exact methods

Many exact methods have been proposed for finding an optimal or a set of optimal solutions for a given problem. Among these methods, we can find branch and bound, branch and cut, and dynamic programming. Due to the inherent combinatorial explosion with respect to the size of the search space for hard COPs in general, this approach is only viable for very small instances. Therefore all practical exact solution approaches try to consider as much of the search space as possible only implicitly, hence ruling out regions where it is guaranteed

that no better feasible solution can be found than a previously found one. Often these methods are based on a tree search, where the search space is recursively partitioned in a divide-and-conquer manner into mutually disjoint subspaces by fixing certain variables or imposing additional constraints. Ruling out regions then amounts to (substantially) pruning the search tree. The scalability of a tree search thus depends essentially on the efficiency of this pruning mechanism. In branch-and-bound (B&B), upper and lower bounds are determined for the objective values of solutions, and subspaces for which the lower bounds exceed the upper bounds are discarded.

I.2.1 Integer programming

This section introduces some basic notations and gives a short introduction into prominent linear programming (LP) and integer programming (IP) techniques. Linear programming is a technique for the optimization of a linear program. More formally, a linear program is an optimization problem in which the objective function and constraints are linear functions of variables. Linear programs which have a feasible solution and are not unbounded always have an optimal solution. For an in-depth coverage of the subject we refer to books on linear optimization [13, 28] as well as on combinatorial and integer optimization [82, 14].

A linear program (LP) is an optimization problem with a linear objective function subject to a set of constraints expressed as linear (in)equalities. A linear program where all the variables are required to be integers is an integer (linear) program (IP). We consider IP problems of the form $Z_{IP} = max\{cx|Ax \leq b, x \geq 0, x \in Z\}$, where c and b are vectors and A is a matrix, where all entries are integers. Further some important classical articles as well as works on current topics regarding IP are given in [60]. We also recommend a more informal paper about linear programming by Dantzig [29]. To process a linear program in continuous variables, the most popular method is the simplex algorithm, which was proposed by Dantzig in 1947, MIP-solvers such as CPLEX [41], etc. One of the most important concepts in integer programming are relaxations, where some or all constraints of a problem are loosened or omitted. Relaxations are mostly used to obtain related, simpler problems that can be solved efficiently yielding bounds and approximate (not necessarily feasible) solutions for the original problem. Embedded within a B&B framework, these techniques may lead to effective exact solution techniques.

$$Z_{LP} = max\{cx | Ax \leq b, x \geq 0, x \in R\}$$

At last, it is said to be a mixed integer program (MIP) if only some variables are restricted to be integer. A mixed integer program (MIP) would involve a combination of integer and real-valued variables and can be written similarly as: $Z_{MIP} = max\{cx + fy | Ax + By \leq b, x, y \geq 0, x \in Z\}$. Maximization problems can be transformed into minimization problems by simply changing the sign of c. In such cases, the linear program is called an integer linear program. Further, if the variables can only take the values 0 or 1, then the corresponding integer linear program is called a binary linear program. Large instances of such LPs can be efficiently solved using simplex-based [27], MIP- solver, etc. Although there exist scenarios where the simplex algorithm, MIP-solvers, etc. show an exponential runtime [65] its average runtime is rather polynomial and it is known to be highly effective in practice.

I.2.2 Dynamic programming

The dynamic programming approach is a useful tool for solving some combinatorial optimization problems. The basic idea was first introduced by Bellman and presented in [12]. This approach consists of:

(1) Breaking a problem up into simpler sub-problems,
(2) Solving these sub-problems,
(3) Combining the sub-solutions to reach the overall solution.

DP is typically applied to optimization problems and following conditions must hold to successfully apply it: (parts of) the sub problems are overlapping, and recursively solving the overall problem in a bottom-up fashion amounts to choosing the right sub problem solutions (i.e. the problem exhibits an optimal substructure). Perhaps the most crucial part is that the sub problems are not disjoint or independent anymore. This fact is exploited via storing their solution's values in some sort of table (or another systematic way) to efficiently retrieve them at the re-occurrence of the sub problems. Hence memory is traded for computational effort. Often the actual solution needs to be reconstructed afterwards, albeit it is usually possible to already derive the required information during the solution process.

I.2.3 Branch and bound method

Branch and bound (B&B) methods are based on the principle of enumerating the solution space of a given problem and then choosing the best solution [72, 77]. B&B is one of the most popular methods to solve optimization problems in an exact manner. The enumeration has a tree structure. Each node of the tree separates the search space into two sub-spaces, until the complete exploration of the solution space [30]. However, there are three aspects in a branch and bound method. They are: (i) Branching strategy, (ii) Bounding strategy and (iii) Node selection strategy. The first branch and bound algorithm for the 0-1 KP was proposed by Kolesar [67]. Several developments have been proposed later [56, 75].

I.2.4 Cutting plane

Gomory [45] proposed the cutting plane algorithm. The principle is to iteratively refine the objective function by adding cuts. A cut can be defined as a constraint that excludes a portion of the search space from consideration. This can reduce the computational efforts in the search process of finding a global optimum solution. In practice it is crucial to have an efficient method for separating cuts as usually a significant number of valid inequalities must be derived until the cutting plane algorithm terminates.

I.2.5 Branch and cut method

The combination of B&B with cutting plane methods yields the highly effective class of branch-and-cut algorithms which are widely used. Specialized branch-and-cut approaches have been described for many applications and are known for their effectiveness. Cut separation is usually applied at each node of the B&B tree to tighten the bounds of the LP relaxation and to exclude infeasible solutions as far as possible. Branch and cut is a method of great interest for solving various combinatorial optimization problems. This method is a result of the integration between two methods:

(1) Cutting plane method,
(2) Branch-and-bound method .

The cutting planes lead to a great reduction in the size of the search tree of a pure branch and bound approach. Therefore, a pure branch and bound approach can be accelerated by the employment of a cutting plane scheme [25, 10, 70].

For small or moderately sized instances exact methods obtain optimal solutions and guarantee their optimality. However, exact methods are unable to solve optimality large instances. This has led researchers to discard exact methods in favour of (meta-)heuristic methods. In fact, (meta-)heuristic methods generate high quality solutions in a reasonable time but there is no guarantee of finding a global optimal solution.

I.3 Metaheuristics approaches

(Meta-)heuristics are a wide class of methods designed to solve approximately many optimization problems. They are approximate algorithms that combine basic heuristic methods into higher level frameworks to efficiently and effectively explore the search space [83]. (Meta-)heuristics are designed to solve complex optimization problems; in fact, the classical heuristics were not always effective and efficient, as they were time consuming or there were some limitation to help them escape from a local optima. of them converge to the optimal solution of some problems with an expected runtime. Several (meta-)heuristic algorithms are studied in the literature such as variable neighborhood search (VNS) [81, 57], tabu search (TS) [46], simulated annealing (SA) [64], genetic algorithm (GA) [47], particle swarm optimization (PSO) [62], among others.

(Meta-)heuristic algorithms based a two strategies [92, 59] : (i) Diversification: that explores the search space to avoid getting stuck in the same or similar areas of feasible space, and (ii) Intensification: that emphasizes on concentrating search in the promising regions previously found, in order to exploiting the potentials.

I.3.1 Simulated annealing

Simulated Annealing is probably one of the first metaheuristics with an explicit strategy to escape from local optima [64]. The basic idea is to allow under some conditions some movements resulting in solutions of worse quality in order to escape from local optima and so to delay the convergence. In fact, at each iteration a random neighbor S_0 of S is

generated and it is accepted as new current solution if its cost function f value is lower than that of the current solution. Otherwise it is accepted with a given probability ρ, this probability of accepting worse solutions decreased during the search process. In fact, the probability of accepting worse solutions is controlled by two factors: the difference of the cost functions and the temperature T. In general, the probability is calculated following the Boltzmann distribution:

$$\rho\big(T, f(S_0), f(S)\big) = Exp\left(-\frac{f(S_0)-f(S)}{T}\right)$$

I.3.2 Variable neighborhood descent

Variable neighborhood descent (VND) is a metaheuristic method proposed in [81] within the framework of variable neighborhood search methods, see [52]. The VND works with k_{max} neighborhood structures N_k, $k = 1, \ldots, k_{max}$, designed for a specific problem. It starts with a given feasible solution as incumbent and sets $k = 1$. If an improvement is obtained within neighborhood N_k, the method updates the new incumbent and sets $k = 1$. Otherwise, it increases the value of k and the next neighborhood is considered. The method stops when a local optimum for $N_{k_{max}}$ is found.

I.3.3 Iterated local search

Iterated Local Search (ILS) framework was defined by Stutzle [97]. An ILS review, its variants, and its applications are detailed in [73]. The idea of iterated local search is very simple. The ILS apply local search to a current solution until a local optimum is reached. In order to overcome this local optimum a perturbation is realized to engender a new starting solution for local search algorithm. The principle of perturbation has a big influence on the process of the ILS method. In fact, if the perturbation is too weak, possibly, the algorithm may not avoid the convergence to the same local optimum. Furthermore, a strong perturbation would change the algorithm to a local search with multi starting solutions.

I.3.4 Variable neighborhood search

Variable neighborhood search (VNS) introduced by Mladenovic and Hansen [81]. VNS is based on the systematic the systemic change within neighborhood structures. In the

beginning of each problem resolution, a set of neighborhood structures $\{N_1, N_2 \dots N_k\}$ of cardinality k must be defined, where $N_k(S)$ the set of solutions in the k^{th} neighborhood of S. Then, from a starting solution S the algorithm increasingly uses complex moves to reach local optima on all selected neighborhood structures. The main steps in VNS algorithm are: shaking, local search and neighborhood move. In the shaking step, a solution S_0 is randomly selected in the k^{th} neighborhood of S. The set of neighborhood structures for shaking phase can be different from the neighborhood structures used in local search. The two well-known search strategies employed as local searches are called first improvement and best improvement. First improvement local search selects the first detected solution S_0 in $N_i(S)$ where S_0 is better than the current solution S. The best improvement method consists in selecting all improving solutions in $N_i(S)$. Many variants are derived from the basic VNS schemes [52]. The well-known are fixed neighborhood search, basic VNS, general VNS, skewed VNS, cyclic VNS, nested VNS, and two-level VNS. These variants indicate that VNS heuristics can be successfully applied to various types of NP-hard optimization problems.

I.4 Cooperatives approaches

The interests about cooperative approaches have grown for the last few years where they have proved their efficiency in solving optimization problems. Since (meta-)heuristics cannot always find the global optimal solution, more and more cooperation schemes between exact methods and (meta-)heuristics are realized. These hybridizations can provide high quality results because they are able to exploit at the same time the advantages of both types of methods.

In the following, we give a brief overview of the three main classifications of cooperative approaches between exact and (meta-)heuristic methods that have been suggested in the literature.

I.4.1 First classification

The Cooperative approaches between exact and (meta-) heuristics were firstly classified in [36, 39] who summarized them into five classes:

(i) Using exact algorithms to explore large neighborhoods within local search algorithms.

(ii) Using information of high quality solutions found in several runs of local search to define smaller problems that are amenable for solution with exact algorithms.

(iii) Exploiting lower bounds in constructive heuristics.

(iv) Using information from integer programming relaxations to guide local search or constructive algorithms.

(v) Using exact algorithms for specific procedures within hybrid (meta)heuristics

I.4.2 Second classification

Puchinger and Raidl [91] have developed the second classification which is divided into two main classes: (i) Collaborative combination and (ii) Integration combination.

(1) Collaborative combination: this class includes hybrid algorithms in which exact algorithm and (meta-) heuristic exchange information, but no algorithm is contained in any other. In this case, both algorithms can be executed in two following cases

i. ***Sequential execution:*** in which one of the algorithms is completely executed before the other. In other words, the (meta-) heuristic algorithm is executed as a preprocessing before the exact method or the (meta-) heuristic algorithm is executed as a post processing after the exact method.

ii. ***Parallel or Intertwined execution***, where both (meta-) heuristic and exact methods are executed in the same time, either in parallel or in an intertwined manner by alternating between both algorithms.

(2) Integration combination: it is termed integrative because when one technique is embedded inside other techniques, in which the first act as a master and the second is seen as a functional component of the first. Obviously, two cases may be considered.

i. The first consists of incorporating an exact algorithm into a (meta-) heuristic. A well-known strategy of this subclass is to solve relaxed problems and to explore large neighborhoods in local search based (meta-)heuristics by means of exact algorithms. Another common strategy is to use an exact algorithm as an operator integrated in evolutionary (meta-)heuristic.

ii. While the second case consists of embedding a (meta-)heuristic within an exact algorithm specially in order to employ (meta-)heuristics to determine incumbent solutions and bounds in branch and bound algorithm.

I.4.3 Third classification

Jourdan et al. [59] are proposed the third classification which can be used to categorize any cooperative algorithm. There are two criteria selected for this classification of cooperation between exact and (meta-) heuristic methods: (i**)** Low-level / high-level, (ii) Relay /teamwork.
(i**)** Low-level / high-level: in this criterion, the hybridization occurs when a given function of an optimization algorithm is replaced by another algorithm. While, in the high level different algorithms are self-contained.
(ii) Relay/ teamwork: when a set of (meta-) heuristics is applied one after another, each one using the solution of the previous one as its inputs, functioning in a pipeline fashion. In the other hand, team hybridization represents a whole cooperation between several optimization models, in which many algorithms, referred as agents, evolve in parallel and each algorithm carries out a search in a solution space. There are four categories that can be derived from this hierarchical classification

Low Level Relay Hybrid (LRH). that corresponds to the cooperative approach wherein a given exact method is embedded into (meta-) heuristic method, or vice-versa. The embedded method is executed sequentially. More precisely, the general method depends on the results obtained by the embedded method. This class of cooperation is frequently used when a (meta) heuristic is used to improve another exact method. For example, to provide a local upper bound associated with each node of the search tree of a branch and bound algorithm, this method can be used to complete the partial solution.

Few examples from literature belong to this category. Augerat et al. [8] developed a LRH cooperation which is based on a branch and cut algorithm (B&C) to solve a capacitated vehicle routing problem (CVRP). The efficiency of the BCA is significantly determinated by the cutting plane generation which is very important issue. They introduce different metaheuristic approaches to extract a set of violated capacity constraints of the relaxed problem.

Low Level Teamwork Hybrid (LTH). contrarily to LRH cooperation, the embedded method is executed in parallel with the general method; with this the performance of the metaheuristics is improved a lot. This hybrid is very popular and has been applied successfully to many optimization problems. Kostikas and Fragakis [69] proposed a cooperative approach to embed a branch and bound algorithm (B&B) into genetic programming (GP). Conventionally, genetic algorithm used recombination operators to generate offspring. An original idea is to incorporate exact method, such as branch and bound algorithm, into recombination operators to find the best offspring from a large set of possibilities.

High Level Relay Hybrid (HRH). In HRH hybrid, numerous self-contained (meta-) heuristics are executed in a sequence. The first case consists in starting (meta-)heuristic approach before an exact algorithm. The (meta-)heuristic approach helps the exact method to speed up the search. The idea consists to use good quality solution found by a (meta-)heuristic as an initial upper bound for B&B method. For example, Klepeis et al. [66] proposed cooperation between B&B algorithm and a conformational space annealing (CSA) to solve the protein structure prediction. HRH cooperation helps to quickly found the active nodes whose lower bound is greater than the upper bound. The second case consists in launching exact algorithm before a (meta-)heuristic approach. Another method consists in using exact algorithm to resolve optimally a relaxed version of the problem under consideration. Then, the obtained solution is exploited to produce initial solution for a (meta)heuristic approach.

High Level Teamwork Hybrid (HTH). As already mentioned, HTH hybrid scheme involves various self-contained metaheuristics performing a search in parallel and cooperating to find an optimum. These various approaches cooperate by exchanging information between them during the search process [24, 18]. In this context, if we consider the cooperation between a branch and bound algorithm and a (meta) heuristic approach the following information may be exchanged:

(i) From a (meta-)heuristic approach to a branch and bound algorithm; the best solution found by the (meta-) heuristic approach is transmitted to branch and bound algorithm in order to help this latter to prune the search tree efficiency. This information is exchanged each time the best solution found is improved.

(ii) From a branch and bound algorithm to a (meta-) heuristic approach;

Nodes of the search tree of branch and bound algorithm with least-cost lower bound represent good partial solutions. The lower bound is used to predict potential interesting search regions. Indeed, these partial solutions are completed and used by heuristic method as initial solutions.

I.5 Matheuristic approach

The cooperative framework between (meta-)heuristics and exact approaches have been performed by many researchers during the last few decades. For instance, Puchinger and Raidl, [91] studied collaborative combinations in which the algorithms exchange information but are not part of each other, and integrative combinations in which one technique is a subordinate embedded component of another technique. For instance, neighborhood search techniques, such as Variable neighborhood descent (VND), Variable Neighborhood Search (VNS) and its variants are proved to be very effective when combined with optimization techniques based on the mathematical programming problem formulations [50]. More precisely, (meta-)heuristic approaches and mathematical programming techniques are two highly successful streams, so it is not surprising that the community tries to exploit and combine the advantages of both. A new subclass of cooperative approaches appeared recently within the term *Matheuristics*. Matheuristics combine (meta-)heuristics and approaches relying on mathematical programming problem formulations. So a number of methods for solving optimization problems which can be considered as matheuristics have emerged over the last decades. Often, exact optimization method is used as the subroutine of the (meta-)heuristics for solving a smaller sub problem [74]. This technique provides interesting results as they take advantages of both types of methods [59]. A classification of algorithms combining local search techniques and exact methods is given in [36, 91]. The focus is particularly on the so called cooperative approaches using exact methods to strengthen local search techniques. They can be viewed as *matheuristics* that combine (meta-)heuristics and mathematical programming [50, 74]. Prandtstetter and Raidl [90] applied a matheuristic that combines an integer linear programming with variable neighborhood search for the car sequencing problem.

I.6 Conclusion

Hard combinatorial problem cannot be solved in an exact way within a reasonable amount of time. Using (meta-)heuristic methods is the most important alternative to solve this class of problems. (Meta-)heuristics approaches are efficient in the search space exploration in a short computation time, but no guarantee about the high-quality solutions. Outlining the advantages and disadvantages of different search techniques we terminate by pointing out the importance of cooperative approaches that can benefit from their advantages while minimizing their drawbacks.

In this thesis, we will be particularly interested in cooperative approaches can be viewed as *matheuristic* that combine neighborhood search techniques and mathematical programming.

The following chapter will be devoted to introduce and solve a new variant and extension of the knapsack problem with setup (KPS) that we call generalized multiple knapsack problem with setup (GMKPS). In fact, in the empirical part of this thesis we will attempt to develop original matheuristics approach to solve GMKPS and its variant MKPS

Chapter II

Cooperative approach for the generalized multiple knapsack problem with setup

II.1 Introduction

In this chapter, we introduce and solve a new variant of knapsack problem with setup (KPS) that we call general multiple knapsack problem with setup (GMKPS). Practical applications of the GMKPS may be seen in production scheduling problems involving setups and machine preferences. A real-life case study of KPS is considered in [21]. It is about a leading manufacturer and supplier of hollow glass in the agro-alimentary glass packing industry, that produces several types of products, including bottles, flacons, and pots with different shapes. To change the production from one product class to another, the production machinery must be setup and molds must be changed in the molding machine. There is no setup between products in the same class. These changes in the manufacturing process require significant setup time and costs. The company operates with a batch delivery policy; products that are manufactured in the same period have the same shipping date**.** Accordingly, the company needs to decide when to make orders so as to maximize the total profit. Hence, the extension of KPS to multiple periods is a real case study of GMKPS. Prices vary according to the customers' expectation of products delivery date; i.e. some customers are willing to pay a higher price for a short lead-time while others are willing to wait for their products in exchange for lower prices. Thus, price, delivery period and total profit have very complex connections that are of extreme interest to businesses today. Thus, we consider that orders could be realized in multiple periods, and the products' price depends on the orders' completion time; i.e. penalties are added to the initial price in case products are not delivered

on-time. In addition, the products (items) could be classified into classes regarding their setups; setup is null between products from the same class. The profit for order j of class i processed in period t is c_{ijt}, and varies for different periods, but the processing time a_{ij} stays the same. To find the assignment of orders that maximizes the total profit, we have to consider the marginal profit of each job, the current production capacity per period, and the setup cost and time from orders. This realistic production scheduling problem is typically our GMKPS case study. Particularly, we deal with multiple knapsack problem with setup (MKPS) if only one setup for each class is allowed during the planning horizon i.e. orders in the same class must be processed in the same period. We note that MKPS is provided in [104], but there is no available benchmark set in the literature.

The GMKPS can be seen as a generalization of classical knapsack problem (KP) [77] where items belong to disjoint classes and can be processed in multiple knapsacks. The selection of a class involves setup costs and resource consumptions (setup time), thus affects both the objective function and the capacity constraint. Note that GMKPS has similarities with several other existing problems in the literature:

- GMKPS is similar to KPS when considering one knapsack [103, 21; 63].
- The MKPS is a special case of GMKPS [104] when items from the same class cannot be assigned to more than a knapsack.
- The multi-item capacitated lot-sizing problem with setup times and shortage costs (MCLSSP) [1] is similar to GMKPS when considering one class of items and the objective is to minimize the total cost induced by the production plan (unit production costs, inventory costs, shortage costs and setup costs).
- The multi-item capacitated lot-sizing problem with times windows and setup times (MCLSP-TW-ST) [38] is similar to GMKPS when considering one class of items and the objective is to minimize the total cost (setup cost, production cost and holding cost).
- The generalized quadratic multiple knapsack problem (GQMKP) [9, 3] is similar to GMKPS, when additional profit is obtained if items j and j' are selected for the same knapsack, and ignoring the setup cost. The maximum number of knapsacks to which the items from the same class can be assigned is a fixed parameter from 1 to the total number of knapsacks.

Other problems exist in literature and seem to have similarities with GMKPS, but they present more differences than similarities:

- The multiple-choice multidimensional knapsack problem (MMKP) [54] is different from the GMKPS. It ignores the setup variables (without y variables), and consists of filling all knapsacks with exactly one item from each class.
- The multiple knapsack problem (MKP) is a special case of MMKP, when considering one class [88].
- The multi-commodity, multi-plant, capacitated facility location problem (denoted, PLANWAR) [102] is required to select the optimum set of plants and warehouses from a potential set and plan production capacities, warehouse capacities and quantities shipped. This problem is different from the GMKPS. It ignores the setup capacity consumption (setup time) and adds the operating cost, where the objective is to minimizing the total operating costs of the distribution network.
- The facility location-allocation problem (FLA) is a particular case of PLANWAR. It ignores the operating costs and consists of defining the best allocation using (α, β)-cost while minimizing the transportation cost [102].

For small and medium sized instances (with less than 10000 variables and 10000 constraints) for similar problems than GMKPS, exact methods such as Branch and bound (Yang, 2006) and Dynamic programming [21] converge to optimality. However, those exact methods are unable to solve large instances in a reasonable time. This has led to discard exact methods in favour of approximated methods such as Multi-start Iterated local search [9] and heuristics based tree search [63]. Nevertheless, metaheuristic methods generate solutions in a reasonable time, but with no guarantee of performance. The purpose of this work is to provide an efficient solving approach for the GMKPS. We introduce a mixed Integer programming (MIP) formulation that, due to the complexity of the GMKPS (more than 60000 variables and 60000 constraints), cannot solve even small test instances (see section 5.3). In fact, it is usually difficult to assign items to the whole sets of knapsacks. In addition, the consideration of the knapsack-dependent cost related to each class of products and the knapsack-dependent profit associated to each item increases the complexity of the problem. Therefore, the design of a new approach providing high quality solutions in a reasonable computing time is quite challenging. An alternative to exact methods would be to develop a matheuristic by combining a meta-heuristic with an exact solving technique: local search techniques to include classes to knapsacks and integer programming (IP) to include items in each knapsack. Our matheuristic approach differs from

existing techniques by the use of the connection between metaheuristic and exact method relying on an effective exploration of the solution space. Experimental results show the performance of the proposed matheuristic on randomly generated instances of GMKPS and its particular case MKPS in comparison to IP: higher quality solution (-0.37% for GMKPS and -0.04% for MKPS) and shorter computation time (20 s vs 3522 s for GMKPS and 11s vs 2965s for MKPS).

The remainder of this chapter is organized as following: In Section II.2, the related literature is presented. Section II.3 contains the mathematical formulations of GMKPS and MKPS. In Section II.4, we propose a matheuristic combining variable neighborhood descent (VND) and integer programming (IP) for GMKPS and MKPS. The experimental results and their interpretations are reported in Section II.5. In Section II.6, we conclude the chapter and give possible and future research ideas.

II.2 Literature review

To deal with the different variants of KP, exact techniques are introduced in the literature. Martello and Toth [76] discussed an upper bound using lagrangian relaxation for MKP. Pisinger [88] presented an exact algorithm using a surrogate relaxation to get an upper bound, and dynamic programming to get the optimal solution. Sinha and Zoltners [95] used two dominance rules for the linear multiple-choice KP to provide an upper bound for the multiple-choice knapsack problem. Chebil and Khemakhem [21] provided an exact method for KPS based on a dynamic program that outperforms the ILP on instances with up to 10,000 items. The time complexity of the dynamic programming grows exponentially with the increasing size of problem. Michel et al. [80] developed an exact method based on a branch and bound algorithm to optimally solve several KPS instances. Yang and Bulfin [103] proposed also exact methods based on a branch-and-bound for KPS, but turns out to solve large instances. Thus, Della et al. [32] suggested an exact approach to optimally solve the 0-1 knapsack problem with setups. The approach relies on an effective exploration of the solution space by exploiting the presence of two levels of variables. It manages to optimality solve all instances with limited computational time. Pferschy and Rosario [87] proposed an exact method based on a dynamic programming motivated by the connection of KPS to a KP with precedence constraints. This pseudo-polynomial algorithm can be stated with fewer variables and constraints and turns out to outperform the recent dynamic programming approach

provided by Chebil and Khemakhem [21]. Moreover, it outperforms the exact approach proposed in Della Croce et al. [32]. The dynamic programming and the Branch and Bound are not practical for solving large problem instances of GMKPS, which is more complex than KPS. Khemakhem and Chebil [63] provided a tree search based combination heuristic for large instances of KPS, but provided less performance results in comparison to dynamic programming. Freville and Plateau [42] provided greedy algorithm and reduction methods for multiple constraints 0-l linear programming problems. Dogan et al. [33] proposed a genetic algorithm solution based approach and Tlili et al. [99] proposed an iterated variable neighborhood descent hyper heuristic for the quadratic multiple knapsack problems (QMKP).

Both exact algorithms and metaheuritics present advantages and drawbacks, when dealing with complex problems, in particular different variants of KPS. The hybridization technique between metaheuristics and exact approaches have been performed by many researchers during the last few decades [91]. This technique provides interesting results as they take advantages of both types of methods [59]. A classification of algorithms combining local search techniques and exact methods is given in [36, 91]. The focus is particularly on the so called hybrid methods using exact methods to strengthen local search techniques. They can be viewed as matheuristics that combine metaheuristics and mathematical programming [50, 24]. Prandtstetter and Raidl [90] applied a matheuristic that combines an integer linear program with variable neighborhood search for the car sequencing problem. Burke et al. [17] studied a hybrid model of Integer Programming and Variable Neighborhood Search for Highly-Constrained Nurse Rostering Problems. Fernandes and Lourenco [39] applied hybrid local search heuristics with exact algorithms to approximately solve different combinatorial optimization problems. Vasquez and Hao [100] proposed a new hybrid approach combining linear programming and tabu search to approximately solve the MKP problem. They considered a two-phased algorithm that first uses Simplex to solve exactly a relaxation of the problem and explores efficiently the solution neighborhood by applying a tabu search approach. Lamghari et al. [71] proposed a hybrid method based on linear programming and variable neighborhood descent for scheduling production in open-pit mines. Adouani et al. [2] applied a matheuristic combining VNS with IP to solve the multiple choice knapsack problem with setup (MCKS) and showed its efficiency for large instances (more than 60000 variables

and 60000 constraints) in comparison to IP with -0.11% as gap of objective value and 13 s vs. 2868 s as difference in computation time.

Local search techniques have proven their efficiency in several combinatorial problems and have been used within hybrid methods for several problems [100, 37, 91]. Particularly, the Variable Neighborhood Descent (VND) is a method based on a systematic change of the neighborhood structures. It is introduced by Mladenović and Hansen [81] and has proven its efficiency on different scheduling problems: unrelated parallel machines with setup times [40], capacitated vehicle routing problem [22], etc.

In this chapter, we propose a new matheuristic approach combining VND and IP (VND&IP) to solve the (G)MKPS. The provided approach relies on an effective exploration of the solution space by exploiting the partitioning of the variables set into two levels. The proposed approach solves approximately, all the instances of (G)MKPS (more than 60000 variables and 60000 constraints) in a limited time in comparison to IP (20 s vs 3522 s for GMKPS and 11s vs 2965s for MKPS). It provides good quality solutions with a negative gap in comparison to IP (-0.37% for GMKPS and -0.04% for MKPS) (see Tables II.4 and II.5 in Section II.5).

II.3 Problem description

We consider a set of T knapsacks each with a capacity b_t, $t \in \{1, \dots, T\}$, and a set of N classes of items. Each class $i \in \{1, \dots, N\}$ consists of n_i items. Let f_{it}, negative integer number ($f_{it} < 0$), denote the setup cost of class i in knapsack t, and d_i, a positive integer number ($d_i > 0$), denote the setup capacity consumption of class i. Each item $j \in \{1, \dots, n_i\}$ of a class i has a profit c_{ijt} (i ∈ {1,...,N}, j ∈ {1,..., n_i}, t ∈ {1,...,T}) and a capacity consumption a_{ij} (i ∈ {1,...,N}, j ∈ {1,..., n_i}). For classes and items assignment to knapsacks, we consider two sets of binary decision variables y_{it} and x_{ijt}, respectively. The variable y_{it} is equal to *1* if knapsack t includes items belonging to class i and *0* otherwise. The variable x_{ijt} is equal to *1* if item j of class i is included in knapsack t and *0* otherwise. We propose the following formulation of the MKPS problem contains $T+T*S+S$ constraints and $T*N+T*S$ variables, where S=$\sum_{i=1}^{N} n_i$:

$$\text{Max Z} = \sum_{\text{t=1}}^{T}\sum_{i=1}^{\text{N}}(f_{it}y_{it} + \sum_{j=1}^{n_i} c_{ijt}x_{ijt}) \quad (1)$$

s.t.

$$\sum_{i=1}^{N}(d_i y_{it} + \sum_{j=1}^{n_i} a_{ij}x_{ijt}) \leq b_t \,; \forall t \in \{1, \dots, T\} \quad (2)$$

$$x_{ijt} \leq y_{it} \,;\ \forall\, i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_i\}, \forall t \in \{1, \dots, T\} \quad (3)$$

$$\sum_{t=1}^{T} x_{ijt} \leq 1 \ ;\ \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_i\} \quad (4)$$

$$x_{ijt}, y_{it} \in \{0,1\} \,;\ \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_i\}, \forall t \in \{1, \dots, T\} \quad (5)$$

Equation (1) represents the objective function that is to maximize the profit of selected items minus the fixed setup costs of selected classes. Constraint (2) guarantees that, for each knapsack $t \in \{1, \dots, T\}$, the sum of the total weight of selected items and the class setup capacity consumption do not exceed the knapsack capacity b_t. Constraint (3) requires that each item is selected only if it belongs to a class that has been setup. Constraint (4) guarantees that each item is selected and assigned to one knapsack at most. Constraint (5) ensures that the decision variables are binary.

The MKPS is a particular case of GMKPS. To get the mathematical formulation for the MKPS, we keep the objective function given in (1), constraints (2), (3) and (5), and replace constraint (4) by constraint (6) because items from the same class cannot be processed in more than one knapsack.

$$\sum_{t=1}^{T} y_{it} \leq 1 \quad \forall i \in \{1, \dots, N\} \quad (6)$$

We note that this mathematical formulation of MKPS contains T+T*S+N constraints and T*N+T*S variables.

Our mathematical modeling of GMKPS can be seen as a generalization to T knapsacks of existing mathematical model for KPS [103, 21, 63, 87], with additional constraint (4). We

note that the mathematical formulation of KPS problem contains 1+S constraints and N+S variables.

Using IP to solve GMKPS and MKPS shows its limitation due to the complexity of the problems (for big instances with up to 60000 variables and 60000 constraints). We show later in the experimental results (Section II.5) that by using IP, only *38* instances (among 360) of MKPS and 7 instances (among 360) of GMKPS are solved to optimality in less than one hour CPU time. For the rest, the computation terminates with an out of memory or is stopped in one hour. Thus, we decided to invest in the development of a matheuristic approach combining VND and IP. We explain our new approach in the next section.

II.4 Matheuristic VND&IP

This chapter contains a new matheuristic combining VND with IP. The main idea of our matheuristic is to decompose the original problem into two problems. The first problem assigns classes to knapsacks (determine the setup variables y_{it}) using a VND that transforms GMKPS (or MKPS) into several independent KPs. Three types of moves have been considered within the VND: *SWAP*, *INSERT* and *DROP/ADD*. The second problem solves each KP using IP (use CPLEX 12.7) that determines the values of x_{ijt} within a very short computation time. For efficiency issue, we apply the IP only if the search space is promising by comparing its result to an upper bound that we provided later. The found values of y_{it} and x_{ijt} yield a feasible solution to GMKPS.

The approach starts with a construction heuristic called reduction-based heuristic (RBH) that provides a good initial solution. Then, three local search procedures (LS_k, $k \in \{1, \dots, 3\}$) are considered within a loop until no further improvement is registered. These local search procedures are obtained by combining each of the movements *SWAP*, *INSERT* and *DROP/ADD* with IP, respectively i.e. LS_1: *SWAP&IP*, LS_2: *INSERT&IP* and LS_3: *DROP/ADD&IP*. Algorithm II.1 shows the whole framework of our matheuristic.

Algorithm II.1: VND&IP

Input: Instance data.
Output: A feasible solution.
Apply the ***RBH heuristic*** to get the initial solution S_{best};
set $k = 1$;
Do
  $S \leftarrow LS_k\ (S_{best})$;
  If $(f(S) > f(S_{best}))$
    $S_{best} \leftarrow S;\ k = 1;$
  Else
    $k = k + 1;$
  EndIf
While $(k < 4)$

In the sequel, we detail the construction heuristic RBH, the calculation of the upper bound for GMKPS[Y_t] that conditions the application of IP after each local search, and the local search techniques *SWAP&IP*, *INSERT&IP*, and *DROP/ADD&IP*.

II.4.1 Initial feasible solution

To generate the initial solution, we use RBH, which is composed of three successive phases:

- **First phase**: We reduce the size of an instance of GMKPS so that all the elements of each class $i \in \{1, \dots, N\}$ are being replaced by a jumbo item (group of items). This item is characterized by a weight a'_i and a profit c'_{it} with $a'_i = \sum_{j=1}^{n_i} a_{ij}$ and $c'_{it} = \sum_{j=1}^{n_i} c_{ijt}\ i \in \{1, \dots, N\}; t \in \{1, \dots, T\}$. In addition to variable y_{it}, we consider variable χ_{it} is equal to 1 if the jumbo item (whole group) of class i is placed in the knapsack t and 0 otherwise. In fact, as the total volume of the item i can exceed the total capacity of a knapsack t, the variables χ_{it} are relaxed i.e. $0 \leq \chi_{it} \leq 1$ (see second phase) and variables y_{it} remain binary to identify the potential classes to be assigned to knapsacks.

  Consequently, the reduced GMKPS (denoted by $GMKPS_{red}$) can be expressed mathematically as follows:

$$\text{Max } Z' = \sum_{t=1}^{T} \sum_{i=1}^{N} (f_{it} y_{it} + c'_{it}\, \chi_{it}) \qquad (7)$$

s.t.

$$\sum_{i=1}^{N} (a'_i \chi_{it} + d_i y_{it}) \leq b_t \qquad t \in \{1, \dots, T\} \qquad (8)$$

$$\chi_{it} \leq y_{it} \; ; \forall i \in \{1, \dots, N\}; t \in \{1, \dots, T\} \qquad (9)$$

$$\sum_{t=1}^{T} \chi_{it} \leq 1 \; ; i \in \{1, \dots, N\} \qquad (10)$$

$$\chi_{it}, \; y_{it} \in \{0,1\}; \; i \in \{1, \dots, N\}; t \in \{1, \dots, T\} \qquad (11)$$

Here, the objective function (7) maximizes the sum of profits related to the selected jumbo items minus the costs induced by the selected classes. The capacity constraint (8) guarantees that the sum of weights for the selected items and classes does not exceed the capacity value b_t . Constraint (9) requires that each item is selected only if it belongs to a class that has been setup. Constraint (10) guarantees that each jumbo item is selected and assigned to one knapsack at most. Constraint (11) ensures that the decision variables y_{it} are binary.

- **Second phase**: It is based on the fixing technique recently proposed by Della et al. [32]. We relax constraint (11) so that $0 \leq \chi_{it} \leq 1$ and $y_{it} \in \{0,1\}$ (only the y_{it} variables are binary). The mixed integer programming for the $GMKPS_{red}$ is optimally solved with CPLEX solver, which provides a feasible combination of y_{it} denoted by 0-1 vector Y_t (t $\in$ {1,…,T}). We construct the set of classes $Y_t = Y_t^1 \cup Y_t^0$, *with* $Y_t^1 = \{i \in \{1, \dots, N\}/y_{it} = 1\}$ and $Y_t^0 = \{i \in \{1, \dots, N\}/y_{it} = 0\}$. Y_t is not guaranteed to be optimal for GMKPS.

  - **Third phase**: once the classes are chosen, GMKPS boils down to several independent standard KPs (denoted GMKPS[Y_t]). Even if KP is known to be weakly NP-Hard, in practice it is well handled by nowadays ILP solvers [61, 78, 77]. Thus, we get the following model for the GMKPS[Y_t] as a sub problem of GMKPS and equivalent to a classical binary KP problem:

  GMKPS[Y_t]:

$$\text{Max } Z_t = \theta_t + \sum_{i \in Y_t^1} \sum_{j=1}^{n_i} c_{ijt} x_{ijt} \qquad (12)$$

s.t.

$$\sum_{i\in Y_t^1}\sum_{j=1}^{n_i} a_{ij}x_{ijt} \leq b'_t \qquad (13)$$

$$x_{ijt} \in \{0,1\}\,; \forall i \in Y_t^1\,; j \in \{1,\dots,n_i\} \qquad (14)$$

where the constants: $\theta_t = \sum_{i\in Y_t^1} f_{it}$, $\gamma_t = \sum_{i\in Y_t^1} d_i \; \forall\, t \in \{1,\dots,T\}$ and $b'_t = b_t - \gamma_t$

Equation (12) represents the objective function for GMKPS[Y_t]. The capacity constraint (13) guarantees that the sum of weights for the selected items does not exceed the capacity value b'_t. Constraint (14) ensures that the decision variables are binary.

We solve each of the $GMKPS[Y_t]$ problems using CPLEX solver. We note S_t the solution of $GMKPS[Y_t]$ and Z_t its profit. We deduce the initial solution for GMKPS $S_{init} = \bigcup_{t=1}^{T} S_t$ and its corresponding profit $Z = \sum_{t=1}^{T} Z_t$. The GMKPS solution is represented by set of variables Y= $\{y_{it}, i = 1,\dots,N;\; t = 1,\dots,T\}$, and a set of variables $X = \{x_{ijt}\,, i = 1,\dots,N;\; j = 1,\dots,n_i;\; t = 1,\dots,T\}$. RBH is applied in the same way when dealing with the MKPS, where the reduction of MKPS ($MKPS_{red}$) is expressed mathematically by equations (7) – (9), (11) and (15):

$$\sum_{t=1}^{T} y_{it} \leq 1\;; i \in \{1,\dots,N\} \qquad (15)$$

In addition to RBH, we consider two other heuristics: Linear Programming based Heuristic (LPH) [105] and Greedy Heuristic (GH) [5]. In our problem, the LPH heuristic is composed of two main phases. We use a CPLEX solver along the procedure of our LPH: In the first phase, we consider again the model GMKPS and remove the integrality constraints on variables x_{ijt} . We limit CPLEX computation time to 500 seconds to obtain an initial solution S_0 . The obtained combination y_{it} is denoted by 0-1 vector $\bar{Y}$. In the second phase, the binary GMKPS $[\bar{Y}]$ is solved to obtain a feasible solution S. The GH heuristic is to build iteratively a feasible solution. In our problem this heuristic is composed of two main phases. In the first phase, the variables y_{it} are fixed randomly. In the second phase, the partial feasible solution obtained in the previous phase is completed by inserting the items one by one to each knapsack t from the set of items that are listed in the decreasing order of their ratio $r_{ijt} = c_{ijt}/a_{ij}$. If the current knapsack is saturated, then

we go to the next knapsack and reapply the two phases on the rest of items, and so on until the saturation of all the knapsacks.

We give in next section an illustration about how the initial solution is obtained using our RBH on a small instance :

Illustration example.

Let consider an example of GMKPS defined by:

$T = 2, N = 4$

$b = [30, 35\ ]$

$n = [3,4,2,3]$

$f = [[-10, -8, -11, -13]; [-9, -10, -11, -9]]$

$d = [5,7,6,4]$

$a = [\{13,10,12\}; \{11,9,13,15\}; \{10,14\}; \{14,15,16\}]$

$\mathrm{C} = \ [[\{\ 13,20,15\}; \{13,15,14,17\}; \{10,17\}; \{16,16,16\}];$
$[\{18,10,15\}; \{13,19,17,15\}; \{10,15\}; \{17,15,16\}]]$

We apply the reduction process to get the $GMKPS_{red}$:

$T = 2, N = 4$

$n = [1,1,1,1]$

$f = [[-10, -8, -11, -13]; [-9, -10, -11, -9]]$

$d = [5,7,6,4]$

$a' = [\ 35,48,24,38]$

$\mathrm{c}' = [[48,59,27,48]; [43,64,25,48]]$

We solve the MIP (only the y_{it} variables are binary) of the $GMKPS_{red}$ and get the following result:

$$Y = \{\ \overbrace{1\,0\,0\,0}^{knapsack\ 1}\quad \overbrace{0\,1\,0\,0}^{knapsack\ 2}\ \}$$

$$X = \{\overbrace{\overbrace{0.7}^{class\ 1}\ \overbrace{0}^{class\ 2}\ \overbrace{0}^{class\ 3}\ \overbrace{0}^{class\ 4}}^{knapsack\ 1}\ \overbrace{\overbrace{0}^{class\ 1}\ \overbrace{0.6}^{class\ 2}\ \overbrace{0}^{class\ 3}\ \overbrace{0}^{class\ 4}}^{knapsack\ 2}\}$$

$Z = 52$.

We can see that the knapsack 1 is set up to accept only items from class 1, and the knapsack 2 is set up to accept only items from class 2. We apply separately:

- GMKPS[Y_1] i.e. IP(1), with a capacity $b[1] - d[1] = 25$, $f[1][1] = -10$.
- GMKPS[Y_2] i.e. IP(2), with a capacity $b[2] - d[2] = 28$, $f[2][2] = -10$.

We obtain the solution X_1 for the GMKPS[Y_1], and the solution X_2 for the GMKPS[Y_2]:

- $X_1 = \{\overbrace{\overbrace{0\ 1\ 1}^{class\ 1}\ \overbrace{0\ 0\ 0\ 0}^{class\ 2}\ \overbrace{0\ 0}^{class\ 3}\ \overbrace{0\ 0\ 0}^{class\ 4}}^{knapsack\ 1}\}$, with $Z_1 = 25$.
- $X_2 = \{\overbrace{\overbrace{0\ 0\ 0}^{class\ 1}\ \overbrace{0\ 1\ 1\ 0}^{class\ 2}\ \overbrace{0\ 0}^{class\ 3}\ \overbrace{0\ 0\ 0}^{class\ 4}}^{knapsack\ 2}\}$, with $Z_2 = 26$.

Thus the initial solution generated by the RBH is:

- $X = \{011000000000000011000000\}$.
- $Z = Z_1 + Z_2 = 51$.

II.4.2 Upper bound for $GMKPS[Y_t]$

Dantzig [31] provided an upper bound for KP. We adapt this upper bound to our problems and provide a new upper bound for the one dimensional knapsack problems $GMKPS[Y_t]$ and $MKPS[Y_t]$. This upper bound is used to decide whether to apply IP or not after the local search in order to explore only fruitful search spaces. It is the same for $GMKPS[Y_t]$ and $MKPS[Y_t]$. We apply the following successive steps to obtain this upper bound:

- **Step1**: Let I denote the set of items of classes $i \in Y_t^1$ sorted in descending order of their efficiency ratio $r_{ijt} = \frac{c_{ijt}}{a_{ij}} \; \forall \, i \in Y_t^1 \; ; \; \forall j \in \{1, \dots, n_i\}$.
- **Step2**: Assign items from I one by one until saturation of the knapsack *t*, i.e., Stop at the first item *i'j'* that cannot be inserted due to capacity saturation of $GMKPS[Y_t]$ (or $MKPS[Y_t]$).
- **Step3**: The upper bound of $GMKPS[Y_t]$ (or $MKPS[Y_t]$) is:

  $UB_t = \theta_t + \sum_{i,j \in I'} c_{ijt} + \frac{b_t - C_t}{a_{i'j'}} c_{i'j't}$, where $C_t = \gamma_t + \sum_{i,j \in I'} a_{ij}$ where *I' is* the set of assigned items and the constants: $\theta_t = \sum_{i \in Y_t^1} f_{it}$ *and* $\gamma_t = \sum_{i \in Y_t^1} d_i$.

II.4.3 SWAP&IP local search

A Swap-based local search requires the definition of a neighborhood structure using simple moves. The considered swap permutes two variables $y_{it} \in Y_t^1$ and $y_{jk} \in Y_k^1$ $(i, j \in \{1, \dots, N\}; t \in \{1, \dots, T-1\}, k \in \{t+1, \dots, T\}$. We change the value of fours setup variables from 1 to 0 and vice versa. A new $GMKPS[Y_t]$ is obtained. In order to save computational effort, before applying IP, we calculate the sum of upper bounds of the new classical knapsacks $GMKPS[Y_t]$ and $GMKPS[Y_k]$ $(UB_{t,k} = UB_t + UB_k)$ and compare it with the total profit of the two knapsacks before Swap move ($Z_{t,k}$=$Z_t + Z_k$). In case $UB_{t,k} > Z_{t,k}$, we apply IP to optimally solve the new classical knapsacks $GMKPS[Y_t]$ and $GMKPS[Y_k]$, respectively, and the best solution is taken as a new initial solution for a next swap process. In case $UB_{t,k} \leq Z_{t,k}$, the search space is not promising as no better solution can be obtained, thus IP is not applied and we proceed to the next step. The procedure is terminated once no improvement is obtained. Algorithm II.2 details the *SWAP&IP* procedure.

Algorithm II.2: $LS_1(S)$

```
Improve ← 1;
S'←S;
While Improve do
    Improve ← 0;
    For t ← 1 to T − 1 do
        n ← card(Y_t^1); // Number of classes in knapsack t.
        For x ← 1 to n do
            i ← Y_t^1[x];
            For k ← t + 1 to T do
```

$m \leftarrow card(Y_k^1)$; // Number of classes in knapsack k.
For $j \leftarrow 1$ *to* m **do** //Swap class i by classes j.
$j \leftarrow Y_k^1[x]$;
$y_{it} \leftarrow 0$; $y_{jk} \leftarrow 0$; $y_{jt} \leftarrow 1$; $y_{ik} \leftarrow 1$;
If $(UB_{t,k} > Z_{t,k})$ **then**
$Z'_{t,k} \leftarrow IP(t) + IP(k)$; //apply IP to solve knapsacks t and k
If $(Z'_{t,k} > Z_{t,k})$ **then**
Store the best S';$Improve \leftarrow 1$;
EndIf
EndIf
EndFor
EndFor
$S \leftarrow S'$; // New starting solution
EndFor
EndFor
EndWhile

II.4.4 INSERT&IP local search

The Insert-based local search is based on a neighborhood search which generates a new solution by removing the class i from knapsack t (change the value of the setup variable $y_{it} \in Y_t^1$ from 1 to 0) and then inserting it into another knapsack k (change the value of the setup variable $y_{it} \in Y_k^0$ from *0* to *1*). We apply IP if ($UB_{t,k} > Z_{t,k}$) by the same way as in the *SWAP&IP* procedure. The best solution is taken as a new initial solution for the next insert-based local search. The procedure is terminated once no improvement is obtained. Algorithm II.3 details the *INSERT&IP* procedure.

Algorithm II.3: $LS_2(S)$

$Improve \leftarrow 1$;
S'←S;
While $Improve$ **do**
$Improve \leftarrow 0$;
For $i \leftarrow 1$ *to* N **do**
$kp \leftarrow \{t \in 1, \dots, T \,/\, y_{it} = 1\}$;
$n \leftarrow card(kp)$; // Number of knapsacks that contain class i
For $x \leftarrow 1$ *to* n **do**
$t \leftarrow kp[x]$;
For $k \leftarrow 1$ *to* T **do**
If ($y_{ik} = 0$) **then**

$y_{it} \leftarrow 0$; $y_{ik} \leftarrow 1$; // Insert i in k and delete it from t
If $(UB_{t,k} > Z_{t,k})$ **then**
$Z'_{t,k} \leftarrow IP(t) + IP(k)$; // apply IP to solve knapsacks t and k
If $(Z'_{t,k} > Z_{t,k})$ **then**
Store the best S'; $Improve \leftarrow 1$;
EndIf
EndIf
EndIf
EndFor
$S \leftarrow S'$; // New starting solution
EndFor
EndFor
EndWhile

II.4.5 DROP/ADD&IP local search

The Drop/Add - based local search is composed of two phases that are applied iteratively: First, Drop and Add moves between setup variables $y_{it} \in \bigcup_{t=1}^{t=T} Y_t^1$ and $y_{jt} \in \bigcap_{t=1}^{t=T} Y_t^0$ are applied. We consider a fictitious knapsack T+1 that contains the non-selected classes. It consists in commuting the value of one variable y_{it} from 1 to 0 (Drop move) then trying to improve the solution using a repair operator (Add move) to change the value of one variable y_{jt} from 0 to 1. Second, the IP is applied if $(UB_t > Z_t)$ to solve the classical knapsacks GMKPS$[Y_t]$. The procedure is terminated once no improvement is obtained. Algorithm II.4 contains the *DROP/ADD&IP* procedure.

Algorithm II.4: $LS_3(S)$

$Improve \leftarrow 1$;

S'←S;
While ($Improve$ **do**
$Improve \leftarrow 0$;
For $t \leftarrow 1$ *to* T **do**
$n \leftarrow card(Y_t^1)$ // Number of classes in knapsack t
For $x \leftarrow 1$ *to* n **do**
$i \leftarrow Y_t^1[x]$;
$Y_{T+1}^1 \leftarrow \bigcap_{t=1}^{t=T} Y_t^0$
// T+1 : Fictitious knapsack that contains the non-selected classes
$m \leftarrow card(Y_{T+1}^1)$; // Number of classes in the Fictitious knapsack T+1
For $x' \leftarrow 1$ *to* m **do** // Swap class i by free classes j
$j \leftarrow Y_{T+1}^1[x']$;
$DROP\ i$ in knapsack t;

```
                    ADD  j   in knapsack t ;
                    If ( UB_t > Z_t )  then
                        Z'_t ← IP(t);
                        If (Z'_t > Z_t) then
                            Store the best S'; // Best solution found
                            Improve ← 1;

                        End if
                    End if
                End for
                S← S'; // New starting solution
            End for
        End for
    End while
```

The *SWAP&IP*, *INSERT&IP*, and *DROP/ADD&IP* procedures are the same for the two problems GMKPS and MKPS.

II.5 Computational experiments

Our approach is implemented and run using C language and CPLEX 12.7 solver on a 2.4 GHZ intel B960 computer with 4 GB of memory.

Due to the unavailability of benchmark instances in the literature, we test our matheuristic *VND&IP* on a set of randomly generated instances of GMKPS and MKPS with a total number of knapsacks T in $\{5, 10, 15,20\}$, all knapsacks are considered small (below the formula of b_t), total number of classes N in $\{10, 20,30\}$, and total number of items n_i for each class i in $[40,60], [60,90]$ and $[90,110]$ (the instances of GMKPS and MKPS are available at the following link: https://goo.gl/zK6yZn).We generate 360 instances in total: *10* instances for each combination (T, N, n_i). We consider the correlation between coefficients by using a random generation scheme that resembles to the ones provided in [21] and [2] which makes use of the following rules:

- The setup cost and capacity consumption are:

$$f_{it} = -\sum_{j=1}^{n_i} c_{ijt} * e$$

$$d_i = \sum_{j=1}^{n_i} a_{ij} * \mathrm{e}$$

Where:

- e is selected with a uniform distribution in $[0.15, 0.25]$.
- a_{ij} is selected with a uniform distribution in $[10, 10000]$.
- $c_{ijt} = a_{ij} + e_1$, where e_1 is selected with a uniform distribution in $[0, 10]$.

$b_t = 0{,}5 * \left(\mathrm{Max}_{1 \le i \le N} \sum_{j=1}^{n_i} a_{ij} + e_2\right)$, where e_2 is selected with a uniform distribution in $[0, \mathrm{Max}_{1 \le i \le N} \sum_{j=1}^{n_i} a_{ij}]$.

Before the experimentation (Section II.5.3), we provide in section II.5.1 a performance analysis of the matheuristic components considering the set of 360 GMKPS instances that are presented in this section II.5. In section II.5.2, we provide a sensitivity analysis regarding the correlations between several coefficients and regarding the knapsacks tightness by applying our approach on a new set of 13 GMKPS instances that are presented in the same section 5.2.

II.5.1 Performance analysis of the VND&IP components

We study here the performance of the main components of our matheuristics, mainly the construction Heuristic RBH and the combination of the three local search techniques *SWAP&IP*, *INSERT&IP* and *DROP/ADD&IP*.

In order to evaluate the performance of RBH, we compare it to GH and LPH heuristics explained in section II.3.1. The RBH, GH and LPH heuristics are tested on all the instances of GMKPS. Table II.1 shows the numerical results on average. The first column contains the name of the heuristic. The second column contains the average of computational time. We note that LPH is stopped at a limit of computation time equal to 500 s. The third column contains the gap between the heuristic solution and the IP solution: $\mathrm{Gap}(\%) = 100 * \left(\frac{\mathrm{IP_{sol}} - \mathrm{Heuristic_{sol}}}{\mathrm{IP_{sol}}}\right)$.

Table II.1. Comparison between RBH, GH and LPH: Average of GMKPS instances.

Heuristic	CPU (s)	Gap (%)
RBH	**0.53**	**1.73 %**
LPH	373.69	3.63 %
GH	0.98	7.032 %

Table II.1 shows that RBH outperforms the other construction heuristics in terms of computation time and solution quality.

We consider the application of our matheuristic with one local search technique (*SWAP&IP*), two local search techniques (*SWAP&IP* and *INSERT&IP*), and three local search techniques (*SWAP&IP*, *INSERT&IP* and *DROP/ADD&IP*). Figure II.1 shows a comparison between these three combinations in terms of average Gap (%) with the IP for the fours instances sets regarding the number of knapsacks. By adding *INSERT&IP*, we observe a slight advantage, for all the set of instances, compared to using only *SWAP&IP*. However, by adding *DROP/ADD&IP*, we observe a higher improvement with a gap that increases when the number of knapsacks increases.

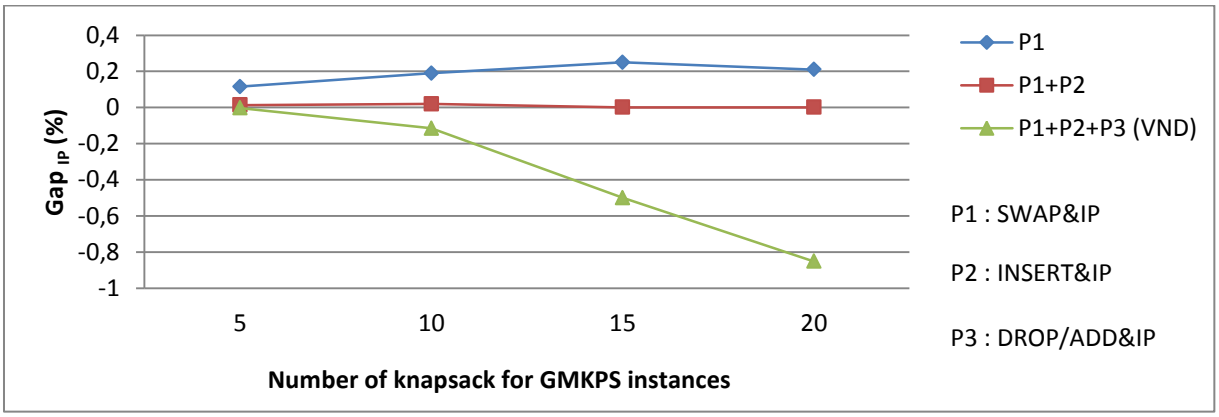

Figure II.1. Effect of VND components

Table II.1 and figure II.1 show that the RBH plays an important role in the overall performance of the provided VND&IP approach: The initial solutions are close to the solutions provided by IP with a gap lower than 1.73%. Figure II.1 shows the efficiency of the other components of VND&IP as they ensure the improvement of the initial solutions provided by the RBH in a very short time. Figure II.1 shows the clear superiority of the VND&IP in comparison to IP, with the contribution of the three local search techniques

SWAP&IP, INSERT&IP and DROP/ADD&IP components respectively equal to 0.19% , 0.01% and -0.37% on average.

II.5.2 Sensitivity analysis of GMKPS parameters

We study here the impact of several parameters values on the complexity of the GMKPS: tightness of knapsacks and correlation between several coefficients.

For the sensitivity analysis regarding the correlation between coefficients, we consider different possibilities:

(1) No correlation between coefficients: $c_{ijt} = v_1$; $a_{ij} = v_2$; $f_{it} = -v_3 * 35$; $d_i = v_4 * 35$; where v_1; v_2; $v_3$and v_4 are uniformly generated in [10, 10000].

(2) Correlation between c_{ijt} and a_{ij} coefficients: $c_{ijt} = a_{ij} + r$; where r is uniformly generated in [0,10] and the other parameters are uncorrelated.

(3) Correlation between c_{ijt} coefficients of the same class: $c_{ijt} = (d_i + r)/100$; where r is uniformly generated in [0,10] and the other parameters are uncorrelated.

(4) Correlation between c_{ijt} and f_{it}: $c_{ijt} = (-f_{it} + r)/100$; where r is uniformly generated in [0,10] and the other parameters are uncorrelated.

(5) Correlation between the a_{ij} coefficients of the same class: $a_{ij} = (d_i + r)/100$; where r is uniformly generated in [0,10] and the other parameters are uncorrelated.

To analyze the impact of these different correlation types, we consider two basic instances with no correlations (1): a small instance with 10 knapsacks and a large instance with 20 knapsacks. Then, we change each instance by including one correlation type at each time and thus generate four additional instances from each basic instance. Tables II.2 reports the numerical results. The first column reports the instance size. The correlation type (from (1) to (5)) is reported in the second column. The notations sol (IP_{sol} and $VND\&IP_{sol}$) and CPU report the solution found and the computational time, respectively. We note that IP is stopped at a limit of computation time equal to one hour. The columns Gap (%) reports the solution gap between IP and VND&IP, calculated as follows: $\text{Gap (\%)} = 100 * \left(\frac{\text{IP}_{\text{sol}} - \text{VND \& IP}_{\text{sol}}}{\text{IP}_{\text{sol}}}\right)$.

Table II.2. Comparison between IP and VND&P for different levels of correlated instances

Instance size	Correlation type	IP		VND&IP		Gap (%)
		IP_{sol}	CPU	$VND\&IP_{sol}$	CPU	
Small	(1)	**197234***	597	197231	43	0.002
	(2)	1271563 -	3600	**1271590**	41	-0.002
	(3)	**2081984** -	3600	**2081984**	55	0.000
	(4)	2321167 -	3051	**2321180**	34	-0.001
	(5)	**3830988***	546	**3830988**	57	0.000
Large	(1)	**163129***	1438	163119	101	0.006
	(2)	4405506 -	3600	**4412796**	139	-0.165
	(3)	8696767-	3600	**8696830**	114	-0.001
	(4)	3690520-	3600	**3701007**	105	-0.284
	(5)	**6732039***	2724	6732000	119	0.001

Best solution in bold, * for optimal solution, and - when IP exceeds the capacity of RAM memory or exceeds the CPU time limit

Table II.2 shows that IP solves to optimality small and large uncorrelated instances i.e. correlation type (1). The VND &IP approach provides good quality solutions for uncorrelated instances in a very short computation time: 43 sec for the small instance (vs 597 sec with IP) and 101sec for the large instance (vs 1438 sec with IP). The IP slightly outperforms the VND&IP when dealing with uncorrelated instances: the gap is equal to 0.002% for the small instance and 0.006% for the large instance. The same phenomenon is observed with correlation type (5).

Table II.2 shows that the IP cannot solve to optimality small and large correlated instances with correlation types (2), (3) and (4); i.e. exceeds the capacity of RAM memory or exceeds the CPU time limit (one hour). Our approach VND& IP solves the instances in very reasonable computational time with an average gap equal to -0.083% for correlated instances type (2), -0.0005% for correlated instances type (3) and -0.142% for correlated instances type (4). We note that the negative gap indicates that VND&IP outperforms IP. The VNS&IP average computation time is 43 sec for small instances and 120 sec for large instances (vs 3600 sec with IP).

We conclude that the correlation of the profits c_{ijt} with other parameters such as weight, setup time, and setup cost, etc. makes the GMKPS more complex to solve i.e. for small and large correlated instances with correlation type (2), (3) and (4), the provided matheuristic VND&IP is more efficient and effective in comparison to IP.

For the sensitivity analysis regarding the tightness of knapsacks, we consider three instances with T = 10, N = 20, n_i in [90, 110] and different sizes of knapsacks (tightness: small, medium and large capacities):

- Small knapsack capacity: $b_t = 0{,}5 * \left(\text{Max}_{1 \le i \le N} \sum_{j=1}^{n_i} a_{ij} + e_2\right)$,
- Medium knapsack capacity: $b_t = 1 * \left(\text{Max}_{1 \le i \le N} \sum_{j=1}^{n_i} a_{ij} + e_2\right)$
- Large knapsack capacity: $b_t = 2 * \left(\text{Max}_{1 \le i \le N} \sum_{j=1}^{n_i} a_{ij} + e_2\right)$

where e_2 is selected with a uniform distribution in $[0, \text{Max}_{1 \le i \le N} \sum_{j=1}^{n_i} a_{ij}]$.

In Table II.3, the first column presents the knapsack tightness of the three instances: small, medium and large knapsacks. The next two columns show the results provided by the IP and *VND&IP*.

Table II.3. Comparison between IP and VND&IP on three instances with different knapsack tightness

knapsack tightness	IP		VND&IP		Gap (%)
	IP_{sol}	CPU	$VND\&IP_{sol}$	CPU	
Small knapsacks	3315613-	3600	**3315697**	147	-0,002
Medium knapsacks	**9437042***	896	9436927	106	0,001
Large knapsacks	**9437176***	84	9437051	123	0,001

Best solution in bold, * for optimal solution, and - when IP exceeds the capacity of RAM memory or exceeds the CPU time limit

Table II.3 shows that IP cannot solve to optimality the instance with small knapsacks. In addition, VND&IP outperforms IP i.e. the gap is negative for the small instance. IP solves to optimality the instances with higher sizes (medium and large knapsacks). From the IP computation time, we can conclude that by decreasing the knapsacks capacities, we increase the GMKPS complexity. From the VND&IP computation time, we can remark that matheuristic is stable with a computation time that does not variate regarding the knapsacks sizes (128 sec on average).

The details of instances used for the sensitivity analysis are available in the following link: https://goo.gl/zK6yZn

II.5.3 Experimentation

Table II.4 summarizes the results obtained by *VND&IP* and IP when solving the GMKPS. Each line presents the average of *10* instances. The first three columns present the number of knapsacks T, the number of classes N and the number of items n_i for each class i. The next three columns show the corresponding average of results provided by the IP, the average of results provided by the matheuristic approach *VND&IP* and the average of the best upper bounds, of all the remaining open nodes in the branch-and-cut tree, provided by CPLEX 12.7 ($CPLEX_{UB}$). The notations sol and CPU report the solution found and the computational time, respectively. We note that IP is stopped at a limit of computation time equal to one hour. Finally, the columns Gap_{IP} and Gap_{UB} report the gap between IP and *VND&IP*, calculated as follows: $Gap_{IP}(\%) = 100 * \left(\frac{\text{IP}_{\text{sol}} - \text{VND \& } IP_{\text{sol}}}{\text{IP}_{\text{sol}}}\right)$, and the gap between CPLEX_{UB} and *VND&IP*, calculated as follows: $Gap_{UB}(\%) = 100 * \left(\frac{\text{CPLEX}_{\text{UB}} - \text{VND\& } IP_{\text{sol}}}{\text{CPLEX}_{\text{UB}}}\right)$, respectively.

Table II.4. Numerical results for GMKPS instances

T	N	n_i	*IP*		*VND & IP*			*UB*	
			IP_{sol}	$CPU(s)$	$VND\&IP_{sol}$	$Gap_{IP}(\%)$	$CPU\ (s)$	$CPLEX_{UB}$	$Gap_{UB}(\%)$
5	10	[40,60]	759992	3342	759994	0.000	5	760026	0.004
		[60,90]	1166555	3600	1166558	0.000	6	1166572	0.001
		[90,110]	1624997	3600	1625002	0.000	3	1625014	0.001
	20	[40,60]	790962	3600	790961	0.000	4	790998	0.005
		[60,90]	1228017	3259	1228019	0.000	4	1228039	0.002
		[90,110]	1616006	3390	1616013	-0.001	5	1616025	0.001
	30	[40,60]	913951	3244	913951	0.000	3	913985	0.004
		[60,90]	114477	2942	1144937	-0.013	3	1144958	0.002
		[90,110]	1703770	3242	1703833	-0.004	4	1703845	0.001
10	10	[40,60]	1445360	3600	1447096	-0.114	23	1447485	0.027
		[60,90]	2364297	3600	2367911	-0.163	15	2368433	0.022
		[90,110]	3083584	3600	3089097	-0.182	17	3089678	0.019
	20	[40,60]	1591134	3600	1591549	-0.029	10	1592221	0.042
		[60,90]	2467619	3600	2472900	-0.224	9	2473423	0.021

		[90,110]	3127235	3168	3131084	-0.123	11	3131550	0.015
	30	[40,60]	1877750	3600	1878104	-0.02	6	1878724	0.033
		[60,90]	2284554	3600	2286305	-0.078	8	2286893	0.026
		[90,110]	3390820	3600	3394182	-0.101	10	3394729	0.016
15	10	[40,60]	2022211	3600	2038186	-0.787	44	2038778	0.029
		[60,90]	3198410	3600	3232685	-1.087	43	3233324	0.020
		[90,110]	4307956	3600	4356089	-1.129	65	4356513	0.010
	20	[40,60]	2281419	3600	2287007	-0.259	26	2287586	0.025
		[60,90]	3659753	3600	3673529	-0.375	23	3674123	0.016
		[90,110]	4762369	3527	4778300	-0.347	23	4778814	0.011
	30	[40,60]	2799889	3600	2801237	-0.048	22	2801916	0.024
		[60,90]	3410638	3600	3419811	-0.283	27	3420318	0.015
		[90,110]	5080972	3600	5089502	-0.174	18	5090030	0.010
220	10	[40,60]	2233779	3600	2257611	-1.068	19	2258214	0.027
		[60,90]	3399647	3600	3424060	-0.723	28	3424686	0.018
		[90,110]	4495295	3600	4538968	-0.973	22	4539506	0.012
	20	[40,60]	2977524	3600	3003851	-0.906	47	3004414	0.019
		[60,90]	4752263	3600	4794759	-0.897	45	4795246	0.010
		[90,110]	6264952	3600	6331424	-1.063	43	6331902	0.008
	30	[40,60]	3661944	3600	3678277	-0.437	16	3678795	0.014
		[60,90]	4463184	3491	4502886	-0.903	41	4503397	0.011
		[90,110]	6679246	3601	6726455	-0.717	36	6727034	0.009

Table II.4 shows that *VND&IP* outperforms IP with a gap on average equal to -0,367%. In detail, the gap on average is -0.002% for $T = 5$, -0.115% for $T = 10$, -0.499% for $T = 15$, and -0.854% for $T = 20$. The CPU on average for *VND&IP* is about 20 s, which is very low in comparison to the average of CPU for IP that is equal to 3522s. For more detailed results, we note that *VND&IP* provides a solution equal to the one provided by the IP for 32 instances (7 optimal and *25* non-optimal) and provides better solutions than IP for 319 instances (see Appendix A). Table II.4 shows that the gap between *VND&IP* and $CPLEX_{UB}$ is 0.015% on average (0.002%, 0.025%, 0.018% and 0.014% for instances with 5, 10, 15 and 20 knapsacks, respectively).

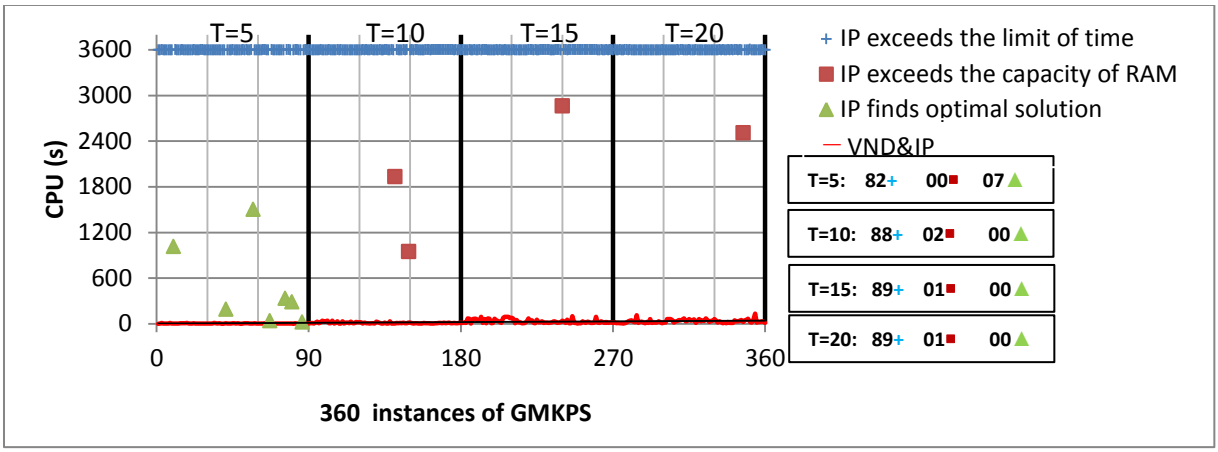

Figure II.2. Computation time of *VND&IP* approach compared to IP for GMKPS

Among the 360 instances of GMKPS, IP finds the optimal solutions for 7 instances, slightly outperforms the *VND&IP* for *9* instances, and for the remaining it terminates with error: exceeds the capacity of RAM memory or exceeds the CPU time limit. Figure II.2 shows the 7 instances solved at optimality are all with $T = 5$.

Table II.5 summarizes the results obtained by *VND&IP* in comparison to those obtained by IP when solving MKPS.

Table II.5. Numerical results for MKPS instances

T	N	n_i	IP		$VND\ \&\ IP$			UB	
			IP_{sol}	CPU(s)	$VND\&IP_{sol}$	$Gap_{IP}(\%)$	CPU(s)	$CPLEX_{UB}$	$Gap_{UB}(\%)$
5	10	[40,60]	337066	2253	337066	0.000	1	337086	0.006
		[60,90]	766837	2310	766842	-0.001	3	766872	0.004
		[90,11	550192	2479	550190	0.000	6	550203	0.002
	20	[40,60]	258448	2965	258448	0.000	2	258469	0.008
		[60,90]	871105	2051	871105	0.000	4	871121	0.002
		[90,11	493255	3241	493256	0.000	6	493267	0.002
	30	[40,60]	421243	1731	421243	0.000	2	421270	0.006
		[60,90]	808314	2104	808315	0.000	17	808331	0.002
		[90,11	559080	2746	559081	0.000	17	559091	0.002
10	10	[40,60]	562993	3204	563000	-0.001	3	563036	0.006
		[60,90]	1480338	2497	1480398	- 0.004	15	1480439	0.003
		[90,11	1029024	3114	1029034	-0.001	5	1029052	0.002

	20	[40,60]	457898	2902	457903	-0.001	5	457941	0.008
		[60,90]	1713710	3170	1713720	-0.001	12	1713756	0.002
		[90,11	938588	2941	938596	-0.001	7	938617	0.002
	30	[40,60]	715887	3600	715893	-0.001	3	715939	0.006
		[60,90]	1551262	3471	1551706	-0.025	12	1551736	0.002
		[90,11	1086450	3510	1086842	-0.040	21	1087103	0.024
15	10	[40,60]	691178	2967	691189	-0.001	5	691248	0.009
		[60,90]	1818377	1971	1818396	-0.001	12	1818426	0.002
		[90,11	1 370342	3083	1370359	-0.001	16	1370653	0.021
	20	[40,60]	658534	3210	658548	-0.002	7	658611	0.010
		[60,90]	2455626	3373	2456683	-0.044	12	2457080	0.016
		[90,11	1250502	3303	1251095	-0.042	16	1251402	0.025
	30	[40,60]	1046226	3600	1047317	-0.107	8	1047381	0.006
		[60,90]	2301745	3603	2302191	-0.019	20	2302725	0.023
		[90,11	1609 642	3444	1613672	-0.199	18	1613929	0.016
220	10	[40,60]	812934	2991	812955	-0.003	5	813014	0.007
		[60,90]	1945837	2465	1945857	-0.001	8	1946348	0.025
		[90,11	1616332	2820	1616357	-0.002	15	1616643	0.018
	20	[40,60]	774016	2380	774284	-0.034	6	774348	0.008
		[60,90]	3051810	3442	3055666	-0.142	23	3056042	0.012
		[90,11	1681049	3339	1686948	-0.368	19	1687326	0.022
	30	[40,60]	1 358070	3487	1359224	-0.084	7	1359279	0.004
		[60,90]	2918388	3575	2922240	-0. 133	22	2922753	0.018
		[90,11	2138531	3602	2146704	-0.353	30	2146999	0.014

Table II.5 shows that *VND&IP* outperforms IP with a gap on average equal to -0.042%. In detail, the gap on average is about 0% for $T = 5$, -0.008 % for $T = 10$, -0.046 % for $T = 15$, and -0.123 % for $T = 20$. The CPU on average for *VND&IP* is about 11 s, which is very low in comparison to the average of CPU for IP that is equal to 2965 s. For more detailed results, we note that *VND&IP* provides a solution equal to the one provided by IP for *76* instances (*38* optimal and *38* non optimal) and provides better solutions than IP for 279 instances (see Appendix A). Table II.5 shows that the gap between *VND&IP* and

$CPLEX_{UB}$ is 0.010% on average. The gap increases when the number of knapsacks increases (0.004%, 0.007%, 0.015% and 0.015% for instances with 5, 10, 15 and 20 knapsacks, respectively).

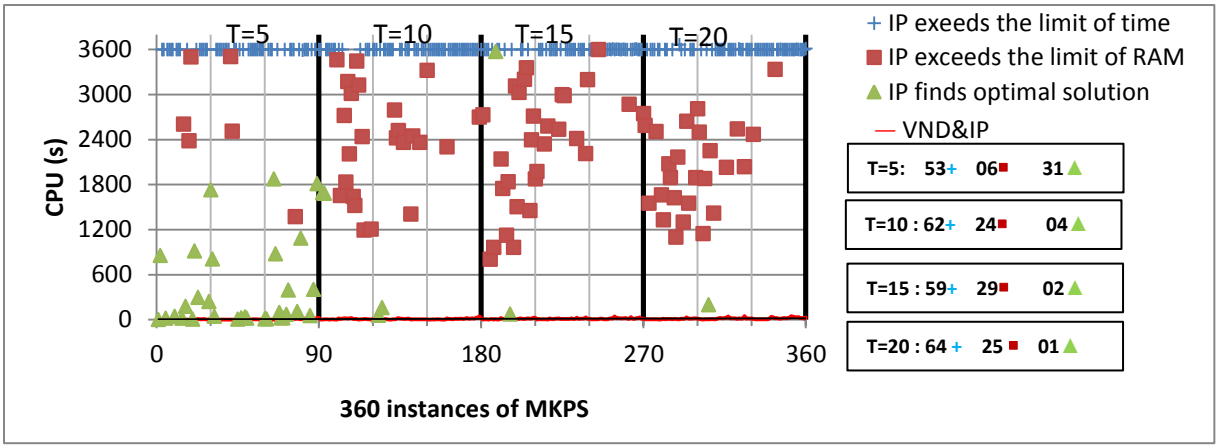

Figure II.3. Solution time of *VND&IP* approach compared to IP for MKPS

Among the 360 instances of MKPS, IP finds the optimal solutions for 38 instances, slightly outperforms the *VND&IP* for 5 instances, and for the remaining it terminates with error: exceeds the capacity of RAM memory or exceeds the CPU time limit. Figure 3 shows that the majority of instances solved at optimality are with $T = 5$ (*31* with $T = 5$, *4* with $T = 10$, *2* with $T = 15$ and *1* with $T = 20$). In addition, we can see that MKPS becomes more difficult when increasing the number of knapsacks *T*. In fact, the number of times that IP terminates with exceeding the capacity of RAM or exceeding the time limit increases from *59* with $T = 5$ to *89* with $T = 20$.

Figure II.4 shows that the results provided by the *VND&IP* approach are better than those provided by the IP for both GMKPS and MKPS. The Gap (%) increases when the number of knapsacks increases. A slight improvement is obtained by our *VND&IP* for MKPS instances (~ -0.04%), and a higher improvement is obtained for the GMKPS instances (~ -0.36%).

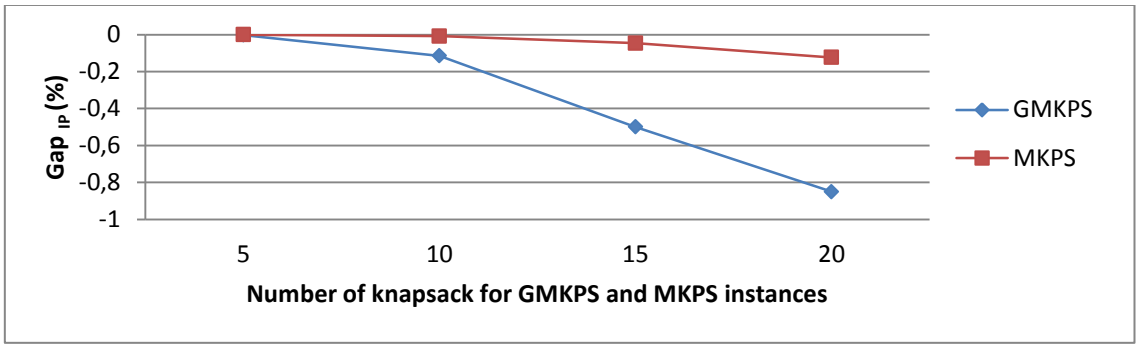

Figure II.4. Quality solution of *VND&IP* for GMKPS and MKPS

The detailed results about the GMKPS and MKPS are available in Appendix A and the following link: https://goo.gl/Knz6Bo.

II.6 Conclusion

This chapter introduces a new variant of the knapsack problem with setup (KPS). We refer to it as the generalized multiple knapsack problem with setup (GMKPS). GMKPS originates from industrial production problems where the items are divided into classes and processed in multiple periods. We refer to the particular case, where items from the same class cannot be processed in more than one period, as the multiple knapsack problem with setup (MKPS). First, we provide mathematical formulations of GMKPS and MKPS and provide an upper bound expression for the knapsack problem. We then propose a matheuristic that combines variable neighborhood descent (VND) with integer programming (IP). We consider local search techniques to assign classes to knapsacks and apply the IP to select the items in each knapsack. Computational experiments on randomly generated instances show the efficiency of our matheuristic in comparison to the direct use of a commercial solver.

For future work, we expect to improve and generalize our matheuristic to deal with other variants of Knapsack problems such as Generalized Quadratic Multiple Knapsack Problem.

Chapter III

Cooperative approach for the Multiple-Choice Knapsack Problem with Setup

III.1 Introduction

The 0-1 Multiple-choice Knapsack Problem with Setup (MCKS) is described as a knapsack problem with additional setup variables discounted both in the objective function and the constraint. Practical applications of the MCKS may be seen in production scheduling problems involving setups and machine preferences. A case study of knapsack problem with setup (KPS) is provided in [32]. To extend the KPS to MCKS, we consider that items from the same family (or class) could be processed in multiple periods.

The MCKS is NP-hard problem, since it is a generalization of the standard knapsack problem (KP) [77]. MCKS reduces to a KP when considering one class, and no setup variables. The KPS is a particular case of MCKS, when the number of period is equal to one (T=1) [7, 21, 63], etc. To the best of our knowledge, Yang [104] is the unique author who dealt with MCKS. He provided an exact method based on a branch and bound for the MCKS, but it has no availability of benchmark instances in the literature. To deal with the different variants of KP, exact techniques are introduced in the literature such as branch and bound algorithm [35, 67], lagrangian decomposition [15], and dynamic programming [87]. Chebil and Khemakhem [21] provided an improved dynamic programming algorithm for KPS. Akinc [6] studied approximated and exact algorithms to solve fixed charge knapsack problem. Michel et al. [80] developed an exact method based on a branch and bound algorithm to solve KPS. Della et al. [32] provided an exact approach for the 0-1 knapsack problem with setups.

Al-Maliky et al. [7] studied a sensitivity analysis of the setup knapsack problem to perturbation of arbitrary profits or weights. Dudzinski and Walukiewicz [35] studied exact methods such as branch-and-bound and dynamic programming for KP and its generalizations. Martello and Toth [76] discussed an upper bound using Lagrangian relaxation for multiple knapsack problem (MKP). Pisinger [88] presented an exact algorithm using a surrogate relaxation to get an upper bound, and dynamic programming to get the optimal solution. Sinha and Zoltners [95] used two dominance rules for the linear multiple-choice KP to provide an upper bound for the multiple-choice knapsack problem.

(Meta-)heuristics approaches have been also developed such as reactive local search techniques [54], tabu search [49], particle swarm optimization [11], genetic algorithm [26], iterated local search [25], etc. Khemakhem and Chebil [63] provided a tree search based combination heuristic for KPS. Freville and Plateau [42] provided a greedy algorithm and reduction methods for multiple constraints 0-l linear programming problem. Tlili et al. [99] proposed an iterated variable neighborhood descent hyper heuristic for the quadratic multiple knapsack problems.

The cooperation technique between exact and (meta-)heuristics approaches have been performed by many researchers during the last few decades. This technique provides interesting results as it takes advantages of both types of approaches [59]. A classifications of algorithms combining local search techniques and exact methods are provided in [36, 91]. The focus in these chapter is particularly on the so called *matheuristic* approach combining local search techniques with integer programming (IP). Fernandes and Lourenco [39] applied cooperative approach to solve different combinatorial optimization problems. Vasquez and Hao [100] proposed a new cooperative approach combining linear programming and tabu search to solve the MKP problem. They considered a two-phased algorithm that first uses Simplex to solve exactly a relaxation of the problem and then explore efficiently the solution neighborhood by applying a tabu search approach. Several works of literature have considered a combination of cooperative approach combining variable neighborhood search with exact technique. Prandtstetter and Raidl [90] applied a cooperative approach that combines an integer linear programming with variable neighborhood search for the car sequencing problem. Burke et al. [17] studied a cooperative approach of Integer Programming and Variable Neighborhood Search for Highly-Constrained Nurse Roistering

Problems. Lamghari et al. [71] proposed a cooperative method based on linear programming and variable neighborhood descent for scheduling production in open-pit mines. To the best of the our knowledge, the combination of VNS with exact technique has never been considered for KPS problem.

The remainder of this chapter is organized as following: Section III.2 contains the mathematical formulations of MCKS. In Section III.3, we propose a cooperative approach can be seen as a *matheuristic* that combine Variable Neighborhood Search (VNS) and integer programming (IP) for MCKS. The experimental results and their interpretations are reported in Section III.4. In Section III.5, we conclude the chapter and give possible and future research ideas

III.2 Problem description

The Multiple Choice Knapsack Problem is defined by knapsack capacity b ∈ N with a set of T divisions (periods), where each division t ∈{1, …,T}, and a set of N classes of items. Each class $i \in \{1, \dots, N\}$ consists of n_i items. Let f_{it}, negative number, de note the setup cost of class i in division t, and let d_i, a positive number, denote the setup capacity consumption of class i. Each item $j \in \{1, \dots, n_i\}$ of a class i has a profit $c_{ijt} \in \mathbb{N}$ and a capacity consumption $a_{ij} \in \mathbb{N}$. For classes and items assignment to divisions of knapsack, we consider two sets of binary decision variables y_{it} and x_{ijt}, respectively. The variable y_{it} is equal to *1* if division t includes items belonging to class i and *0* otherwise. The variable x_{ijt} is equal to *1* if item j of class i is included in division t and *0* otherwise. We propose the following mathematical formulation for the MCKS:

$$\text{Max z} = \sum_{\text{t=1}}^{T} \sum_{i=1}^{\text{N}} (f_{it} y_{it} + \sum_{j=1}^{n_i} c_{ijt} x_{ijt}) \qquad (1)$$

$$\sum_{t=1}^{T} \sum_{i=1}^{N} (d_i y_{it} + \sum_{j=1}^{n_i} a_{ij} x_{ijt}) \leq b \qquad (2)$$

$$x_{ijt} \leq y_{it} \; ; \; \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_i\}, \forall t \in \{1, \dots, T\} \qquad (3)$$

$$\sum_{t=1}^{T} x_{ijt} \leq 1 \ ; \ \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_i\} \quad (4)$$

$$x_{ijt}, y_{it} \in \{0,1\} \ ; \ \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_i\}, \forall t \in \{1, \dots, T\} \quad (5)$$

Equation (1) represents the objective function that is to maximize the profit of selected items minus the fixed setup costs of selected classes. Constraint (2) guarantees that the sum of the total weight of selected items and the class setup capacity consumption does not exceed the knapsack capacity b. Constraint (3) requires that each item is selected only if it belongs to a class that has been setup. Constraint (4) guarantees that each item is selected and assigned to one division at most. Constraint (5) ensures that the decision variables are binary.

Using CPLEX 12.7 to solve MCKS shows its limitation due to the complexity of the problems. We show later in the experimental results (Section II.4) that by using CPLEX, only *27* instances of MCKS among 120 are solved to the optimality in less than 1 h CPU time. For the rest, the computation terminates with an out of memory or is stopped at 1 h. Thus we decided to invest in the development of a cooperative approach can be seen as a *matheuristic* combining variable neighborhood search and integer programming (VNS&IP). We explain our new approach in the next section.

III.3 Matheuristic approach for MCKS

Local search techniques have proven their efficiency in several combinatorial problems and have been used within cooperative approaches for several problems [36, 33]. Particularly, the Variable Neighborhood Search (VNS) is a method based on a systematic change of the neighborhood structures. It is introduced by Maldenovic and Hansen [81] and has proven its efficiency on different scheduling problems: location routing [57] car sequencing problem [90], for the recent surveys on VNS see [51, 52].

This chapter contains a new matheuristc approach combining VNS with IP. The main idea of our cooperative approach is to decompose the original problem in to two sub-problems (two levels). The first problem (first level) is to assign classes to the divisions of knapsack (determine the setup variables y_{it}) using a VNS approach allowing the transformation of

MCKS into classical KP. Two movements have been considered within the VNS approach: local search procedure (*LS)* and a perturbation mechanism which represents the core idea of VNS, is applied to y_{it} *variables*. The perturbation phase aims to change the neighborhoods structure, N_k), when the algorithm is trapped at a local optimum. The second problem (second level) is to solve the classical KP by considering the IP that determines the values of x_{ijt} with a very short computation time. For efficiency issue, we apply the IP only if the search space is identified to be promising by comparing its result to an upper bound that we provided later. Note the found values of y_{it} and x_{ijt} yield a feasible solution to MCKS.

The approach starts with a construction heuristic called reduction-based heuristic (RBH). Then, the obtain solution S_0 is improved by using a Local search technique with integer programming (LS&IP) procedure. At each iteration, Perturb&IP and LS&IP are successively applied to the best current solution S. More precisely, A set of neighborhoods N_k, $k = 1, \dots, kmax$ is initialized. At each iteration the perturbation mechanism PERTURB&IP is applied based on neighborhood N_k to obtain new solution S_2, then the two local search SWAP&IP and INSERT&IP are applied to obtain a new solution S_3. If this new solution is better than S, then this latter is updated and the process continues with the first neighborhood N_1 (S), otherwise the same steps are repeated with the next neighborhood N_{k+1}. The algorithm works until a termination condition is satisfied. Algorithm III.1 shows the whole framework of our approach.

Algorithm III.1 : VNS&IP ($Data, t\_max,$ k_{max} **)**

$S_0 \leftarrow$ **RBH**(Data);
$N_k, k = 1, \dots, k_{max}$ /*neighborhood structure */
$S_1 \leftarrow$ **LS&IP**(S_0); /*Local search*/
$S \leftarrow S_0$
While $(t < t\_max)$
  $k \leftarrow 1$;
  Do
    $S_2 \leftarrow$ **PERTURB&IP (**N_k, S_1**)** /* Random neighbor*/
    $S_3 \leftarrow$ **LS&IP**(S_2)
    If (S_3 > S **) then** S$\leftarrow S_3$; k$\leftarrow$1;
    else k$\leftarrow$ k+1**;**
  While $(k=k_{max})$
 $t \leftarrow CPU\_time()$;
End while
return S ;

In the sequel, we detail the construction heuristic RBH, the calculation of the upper bound for IP that condition the application of IP after each local search move.

III.3.1 Initial feasible solution

To generate the initial solution, we adapt and extend a construction heuristic is based on a reduction based heuristic (RBH) recently proposed in chapter II. For illustration, we considered the MCKS problem and explain below the three successive phases of our RBH:

- First phase: We reduced the MCKS so that every class contains a single object ($n_i = 1, i \in \{1, \dots, N\}$). This object is characterized by a weight a'_i and a profit c'_{it} with $a'_i = \sum_{j=1}^{n_i} a_{ij}$ and $c'_{it} = \sum_{j=1}^{n_i} c_{ijt}\ i \in \{1, \dots, N\}; t \in \{1, \dots, T\}$. Consequently, the reduced MCKS ($MCKS_{red}$) can be expressed mathematically as follows:

$$\text{Max } z' = \sum_{t=1}^{T} \sum_{i=1}^{N} (c'_{it} x_{it} + f_{it} y_{it}) \quad (6)$$

s.t.

$$\sum_{t=1}^{T} \sum_{i=1}^{N} (a'_i x_{it} + d_i y_{it}) \leq b \quad ; i \in \{1, \dots, N\}; t \in \{1, \dots, T\} \quad (7)$$

$$0 \leq x_{it} \leq y_{it}\ ;\ \forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, n_i\}, \forall t \in \{1, \dots, T\} \quad (8)$$

$$\sum_{t=1}^{T} x_{it} \leq 1\ ; i \in \{1, \dots, N\}; t \in \{1, \dots, T\} \quad (9)$$

$$y_{it} \in \{0,1\};\ i \in \{1, \dots, N\}; t \in \{1, \dots, T\} \quad (10)$$

- Second phase: we relaxed constraint (8) so that $0 \leq x_{it} \leq 1$ and $y_{it} \in \{0,1\}$. The relaxed model of $MCKS_{red}$ is solved using IP, which gives the values of y_{it}. We constructed the set of classes $Y_t = Y_t^1 \cup Y_t^0$, $with\ Y_t^1 = \{i \in \{1, \dots, N\}/y_{it} = 1\}$ and $Y_t^0 = \{i \in \{1, \dots, N\}/y_{it} = 0\}$.

- Third phase: We considered the following IP for the MCKS[Y] as a KP problem:

$$\text{IP: Max } Z_t = \sum_{t=1}^{\mathrm{T}} \sum_{i \in Y_t^1} \sum_{j=1}^{n_i} c_{ijt} x_{ijt} + \theta_t \qquad (11)$$

s.t.

$$\sum_{t=1}^{\mathrm{T}} \sum_{\mathrm{i} \in Y_t^1} \sum_{j=1}^{\mathrm{ni}} a_{ij} x_{ijt} \leq b - \gamma \qquad (12)$$

$$x_{ijt} \in \{0,1\}\,; \forall i \in Y_t^1\,; j \in \{1, \dots, n_i\} \qquad (13)$$

Where $\theta_t = \sum_{\mathrm{i} \in Y_t^1} f_{it}$, and $\gamma = \sum_{\mathrm{i} \in Y_t^1} d_i \quad \forall\, t \in \{1, \dots, T\}$.

We solved the $MCKS[Y]$ problems and noted also IP using CPLEX solver. The MCKS solution is represented by set of variables Y= $\{y_{it}, i = 1, \dots, N;\ t = 1, \dots, T\}$, and a set of variables $X = \{x_{ijt}\,, i = 1, \dots, N;\ j = 1, \dots, n_i; t = 1, \dots, T\}$.

In addition to RBH, we considered two other heuristics: Linear Programming based Heuristic (LPH) [48, 105] and Greedy Heuristic (GH) [93, 5]. In our problem the LPH heuristic is composed of two main phases: In the first phase, the relaxation of the MCKS (binary y_{it} and continues variables x_{ijt}) is solved to determine the variables y_{it}. In the second phase, the reduced MCKS is solved by using CPLEX solver to determine the variables x_{ijt}. The GH heuristic is to build iteratively a feasible solution. In our problem this heuristic is composed of two main phases. In the first phase, the variables y_{it} are fixed randomly. In the second phase, the partial feasible solution obtained in the previous phase is completed by inserting the items one by one until saturation of the knapsack from the set of items that are listed in the decreasing order of their ratio $r_{ijt} = c_{ijt}/a_{ij}$.

III.3.2 Upper bound for IP

Dantzig [31] provided an upper bound for KP. We adapted this upper bound to our problem and provided a new upper bound for each division t of MCKS. This upper bound was used to decide whether to apply IP or not after the local search in order to explore only fruit full search spaces. We applied the following successive steps to obtain this upper bound:

- **Step1**: Let I denote the set of items of classes $i \in Y_t^1$ sorted in descending order of their efficiency ratio $r_{ijt} = \frac{c_{ijt}}{a_{ij}}$ $\forall\, i \in Y_t^1$; $\forall j \in \{1, \dots, n_i\}$.
- **Step2**: Assign items from I one by one until saturation of the knapsack, i.e., Stop at item *i'j'* that cannot be inserted due to capacity saturation of $MCKS[Y]$.
- **Step3**: The upper bound of division t is:

$UB_t = \sum_{i \in Y_t^1} f_{it} + \sum_{i,j \in I'} c_{ijt} + \frac{b-C}{a_{i'j'}} c_{i'j't}$, where $C = \sum_{t=1}^{T} (\sum_{i \in Y_t^1} d_i + \sum_{i,j \in I'} a_{ij})$ with I' the set of assigned items, and (b – C) the residual capacity for division t .

III.3.3 Local search with IP

In the local search phase, two neighborhood structures, SWAP&IP and INSERT&IP operators are employed with in the Algorithm III.2 .

Algorithm III.2: LS&IP (data**,** S_{best})
Input: Instance data & best solution found
Output: A feasible solution S''
/* S_{best}: best solution found by **RBH** (first iteration) or by **perturb&IP** */
Do
$improve \leftarrow 0$;
$S1 \leftarrow SWAP\&IP\ (S_{best})$
$S2 \leftarrow INSERT\&IP\ (S1)$
If $(f(S2) > f(S_{best}))$
$S_{best} \leftarrow S2$; $improve \leftarrow 1$;
EndIf
While $(\boldsymbol{improve == 1})$
Return S_{best}

SWAP&IP. A Swap-based local search requires the definition of a neighborhood structure using simple moves so as to produce a set of neighbor solutions which permits to explore more search spaces and thus provide high quality solutions. The considered swap process consists of permuting two variables $y_{it} \in Y_t^1$ and $y_{jk} \in Y_k^1$ $(i, j \in \{1, \dots, N\}; t \in \{1, \dots, T\}, k \in \{t+1, \dots, T+1\}$. where *T+1* is a fictive knapsack that contains all the non-selected classes. We changed the value of setup variables from 1 to 0 and vice versa. A new $MCKS[Y]$ was obtained. In order to save computational effort, before applying IP, we calculated the sum of upper bounds of the new divisions t and k ($UB_{t,k} = UB_t + UB_k$) and compared it with the

total profit of the two divisions before Swap move ($Z_{t,k}$=$Z_t + Z_k$). In case $UB_{t,k}$>$Z_{t,k}$. We applied IP to optimally solve the new classical knapsack $MCKS[Y]$ and the best solution was taken as a new initial solution for a next swap process. In case $UB_{t,k} \leq Z_{t,k}$, the search space was not promising as no better solution could be obtained, thus IP(t, k) was not applied and we proceeded to the next step. The procedure is terminated once no improvement is obtained. Algorithm III.3 details the *SWAP&IP* procedure.

Algorithm III.3 : SWAP&IP (data, S ***)***

Input: Instance data & initial solution
Output: A feasible solution

```
do
  Improve ← 0;
    For t ← 1 to T do
        n ← card(Y_t^1); /* Number of classes in division t */
        For x ← 1 to n do
            i ← Y_t^1[x];
            For k ← t + 1 to T + 1 do
                m ← card(Y_k^1); /* Number of classes in division k */
                For j ← 1 to m do /* Swap class i by each class j */
                    j ← Y_k^1[x];
                    y_it ← 0; y_jk ← 0; y_jt ← 1; y_ik ← 1;
                    If (UB_t,k > Z_t,k) then
                      Z'_t,k ← IP(t, k);
                      If (Z'_t,k > Z_t,k) then
                          Store the best S';Improve ← 1;
                      EndIf
                    EndIf
                EndFor
            EndFor
            S ← S'; /* New starting solution */
        EndFor
    EndFor
  While (improve==1)
Return S
```

INSERT&IP. The Insert-based local search is based on a neighborhood search which generates a new solution by removing the class i from knapsack t (change the value of the setup variable $y_{it} \in Y_t^1$ from 1 to 0) and then inserting it into another knapsack k, $k \in \{1, \dots, T+1\}$, The IP(t, k) is applied if ($UB_{t,k} > Z_{t,k}$) by the same way than in the *SWAP&IP* procedure. The best solution is taken as a new initial solution for the next insert-based local

search. The procedure is terminated once no improvement is obtained. Algorithm III.4 details the *INSERT&IP* procedure.

Algorithm III.4: INSERT&IP(data, S)

Input: Instance data & initial solution
Output: A feasible solution
do
  $Improve \leftarrow 0$;
  For $i \leftarrow 1$ *to* N **do**
    $kp \leftarrow \{t \in 1, \dots, T \,/\, y_{it} = 1\}$;
    $n \leftarrow card(kp)$; /* Number of divisons that contain class i */
    For $x \leftarrow 1$ *to* n **do**
      $t \leftarrow kp[x]$;
      For $k \leftarrow 1$ *to* $T + 1$ **do**
        If ($y_{ik} = 0$) **then**
          $y_{it} \leftarrow 0$; $y_{ik} \leftarrow 1$; /* Insert i in k and delete it from t */
          If $\left(UB_{t,k} > Z_{t,k}\right)$ **then**
            $Z'_{t,k} \leftarrow IP(t) + IP(k)$;
            If $(Z'_{t,k} > Z_{t,k})$ **then**
              Store the best S'; $Improve \leftarrow 1$;
            EndIf
          EndIf
        EndIf
      EndFor
    $S \leftarrow S'$; /* New starting solution */
    EndFor
  EndFor
While (improve ==1)
Return S

PERTURB & IP. The design of the perturbation mechanism is crucial for the performance of the algorithm. If the mechanism provides too small perturbation, local search may return to the previously visited local optimum points and no further improvement can be obtained. The mechanism consists of strongly perturbing a part of the current solution to jump the local optima and obtain a new starting solution. Two phases were applied iteratively in order to simulate this jumping principle: The first is a select of k randomly chosen items (setup variables y_{it}) and the second is the IP which is applied to solve the classical knapsacks $MCKS[Y]$. The resulting solution is accepted according to the following condition if ($(f(s') > \mathfrak{y}\, f(s))$, where $\mathfrak{y}$ that is constant value between 0 and 1. The perturbation

method was terminated when the total number of applied moves (perturbation length) equals to the $\mathbf{p_{max}}$. Algorithm III.5 provides a description of the new local search method.

Algorithm III.5: **Perturb&IP(data, S)**

Input: Instance data & best solution found S
Output: A randomly feasible solution
$p \leftarrow 1$;
k←Number of selected classes in best solution S
Do
Select a random set of k classes , from N
Randomly assigned the y_{it}variables
Apply IP to fix the x_{ijt} variables
If $(f(s') > \mathfrak{y}\, f(s))$ **then**
Store the best S' /* best solution found */
$p \leftarrow 1$;
Else
p←p+1;
End if
while $p \leq \mathbf{p_{max}}$
return S'

III.4 Computational results

For computation, our approach was implemented and run using C language and CPLEX 12.7 solver on a 2.4 GHZ intel B960 computer with 4 GB of memory. Due to the unavailability of benchmark instances in the literature, we tested our cooperative approach *VNS&IP* on a set of randomly generated instances of MCKS with a total number of periods T in $\{5, 10, 15, 20\}$, total number of classes N in $\{10, 20, 30\}$, and total number of items n_i for each class i in [90,110] (Available at https://goo.gl/4fz6fg). We generated *120* instances in total: *10* instances for each combination (T, N). We designed a random generation scheme, as presented in [21], where:

- a_{ij} is selected with a uniform distribution in [10,10000].
- $c_{ijt} = a_{ij} + e_1$, $e_1$is selected with a uniform distribution in[0,10].

- $b = 0{,}5 * \sum_{i=1}^{N} \sum_{j=1}^{n_i} a_{ij}$.
- $d_i = \sum_{j=1}^{n_i} a_{ij} * \mathrm{e}$.
- $f_{it} = -\sum_{\mathrm{j=1}}^{\mathrm{n_i}} \mathrm{c_{ijt}} * \mathrm{e}$, e is selected with a uniform distribution in[0.15,0.25].

The Gap report the standard deviation between IP and VNS&IP that is calculated as follows: $\mathrm{Gap}(\%) = 100 * \left(\frac{\mathrm{IP_{sol}} - \mathrm{VNS\ \&\ } IP_{\mathrm{sol}}}{\mathrm{IP_{sol}}}\right)$.

III.4.1 Parameter setting

Generally, when using approximate algorithms to solve optimization problems, it is well known that different parameter settings for the approach lead to different quality results. The parameters for VNS&IP are as follows: $\mathbf{time_{max}}$, the maximal time measured in seconds and its fixed to T, where T is the number of periods (divisions). $\mathbf{k_{max}}$, the maximum number of consecutive failed iterations is fixed to N, where N is the number of classes. The perturbation length $\mathbf{p_{max}}$ is fixed to T. $\mathfrak{y}$ that is constant value between 0 and 1 to relax the acceptance condition is fixed to 0.8. It is worth pointing out that a different adjustment of method's parameters would give important findings. But this better adjustment would sometimes lead to heavier execution time requirements. The set of values chosen in our experiment represents a satisfactory trade-off between quality solution and running time.

III.4.2 Computational results

Before the experimentation, the effect on performance of the main components of our algorithm is assessed, mainly the construction Heuristic RBH and the combination of the two local search techniques *LS&IP* and *PERTURB&IP*.

In order to evaluate the performance of RBH, we compared it to HG and LPH heuristics explained in section III.3.1. The RBH, HG and LPH heuristics are tested on all the instances of MCKS. Table III.1 shows the numerical results on average. The first column contains the name of the heuristic. The second column contains the average of computational time. We noted that LPH is stopped at a limit of computation time equal to 500 s. The third

column contains the gap between the heuristic solution and the IP solution: $\text{Gap}(\%) = 100 * \left(\frac{\text{CPLEX}_{\text{sol}} - \text{Heuristic}_{\text{sol}}}{\text{CPLEX}_{\text{sol}}}\right)$.

Table III.1. Comparison between RBH, HG and LPH: Average of MCKS instances.

Heuristic	CPU (s)	Gap (%)
RBH	**0.63**	**1.46 %**
LPH	304	5.34 %
GH	0.51	8.13 %

Table III.1 shows that RBH outperforms the other construction heuristics in terms of computation time and quality solution.

It is important to give information about the impact of the LS&IP and PERTURB&IP on the performance of VNS&IP. We consider the application of our cooperative approach with RBH, RBH+LS&IP and RBH+LS&IP+PERTURB&IP (VNS&IP). Table III.2 shows a comparison between these three combinations in terms of average Gap (%) with the IP for the four set instances regarding the number of periods (divisions). Each line presents the average of 10 instances. The first two columns present the number of divisions (or periods) T and the number of classes N . The next three columns show the corresponding average gap between **RBH** and IP, the average gap between **RBH+LS&IP** and IP, and the average gap between **RBH+LS&IP+PERTURB&IP** (VNS&IP) and IP.

$$\text{Gap}(\%) = 100 * \left(\frac{\text{IP}_{\text{sol}} - \text{Heuristic}_{\text{sol}}}{\text{IP}_{\text{sol}}}\right).$$

Table III.2: Effect of VNS&IP components

Instances		**RBH**	**RBH + LS&IP**	**RBH + LS&IP + PERTURB&IP**
T	**N**			
5	**30**	0,95	0,23	-0,054
10		1,23	0,39	-0,012
15		1,82	0,45	-0,125
20		1,99	0,42	-0,155

Table III.2 shows that by adding LS&IP, we observe an important advantage, for all the set of instances, compared to using only RBH. However, by adding PERTURB&IP, we

observe a higher improvement with a gap that increases when the number of knapsacks increases. For the experimentations below, we considered the best combination with RBH as construction heuristic, LS&IP as local search techniques and PERTURB&IP as perturbation mechanism.

Table III.3 summarizes the results obtained by *VNS&IP* and IP when solving the MCKS. Each line presents the average of *10* instances. The first two columns present the number of divisions (or periods) T and the number of classes N. The next three columns show the corresponding average of results provided by CPLEX, the average of results provided by the cooperative approach *VNS&IP* and the average of the best upper bounds, of all the remaining open nodes in the branch-and-cut tree, provided by CPLEX 12.7 ($CPLEX_{UB}$). The notations sol and CPU report the solution found and the computational time, respectively. We note that CPLEX is stopped at a limit of computation time equal to 1h. Finally, the columns Gap_{IP} and Gap_{UB} report the gap between CPLEX and VNS&IP, calculated as follows: $Gap_{IP}(\%) = 100 * \left(\frac{\text{CPLEX}_{\text{sol}} - \text{VNS \& } IP_{\text{sol}}}{\text{CPLEX}_{\text{sol}}}\right)$, and the gap between CPLEX_{UB} and *VNS&IP*, calculated as follows: $Gap_{UB}(\%) = 100 * \left(\frac{\text{CPLEX}_{\text{UB}} - \text{VNS\& } IP_{\text{sol}}}{\text{CPLEX}_{\text{UB}}}\right)$, respectively.

Table III.3. Numerical results for MCKS instances.

T	N	*CPLEX*		*VNS & IP*			*UB*	
		$\text{CPLEX}_{\text{obj}}$	*CPU*	$\text{VNS\& } IP_{\text{sol}}$	*CPU*	$Gap_{IP}(\%)$	$CPLEX_{UB}$	$Gap_{UB}(\%)$
5	10	1772249	1735	1773409	6	-0,066	1773431	0,001
	20	3571514	2863	3573719	6	-0,063	3573771	0,001
	30	5398429	2267	5401333	6	-0,054	5401369	0,001
10	10	1795187	2587	1795188	11	0,000	1795221	0,002
	20	3602956	3439	3603067	10	-0,003	3603090	0,001
	30	5445060	2937	5445715	11	-0,012	5445752	0,001
15	10	1793209	2819	1795262	15	-0,118	1795311	0,003
	20	3605797	3333	3617045	15	-0,315	3617079	0,001
	30	5471310	3255	5478013	15	-0,125	5478052	0,001
20	10	1793091	2745	1796768	20	-0,208	1796796	0,002

20	3609105	3481	3615497	20	-0,180	3615547	0,001
30	5454676	2961	5463066	20	-0,155	5463115	0,001

Table III.3 shows that *VNS&IP* outperforms IP with a gap on average equal to -0.11%. In detail, the gap on average is about -0,06% for $T = 5$, -0.005 % for $T = 10$, -0.18 % for $T = 15$, and -0.18 % for $T = 20$. The CPU on average for *VNS&IP* is about 13 s, which is very low in comparison to the average of CPU for CPLEX that is equal to 2868 s. For more detailed results, we note that *VNS&IP* provides a solution equal to the one provided by CPLEX for *51* instances and provides better solutions than CPLEX for 65 instances (see, Appendix A or https://goo.gl/w44aUs). Table III.2 shows that the gap between *VNS&IP* and $CPLEX_{UB}$ is 0.001% on average.

Among the 120 instances of MCKS, CPLEX finds the optimal solutions for 27 instances, slightly outperforms the *VNS&IP* for 4 instances, and for the remaining it terminates with error: exceeds the capacity of RAM memory or exceeds the CPU time limit. the majority of instances solved at optimality are with $T = 5$ (*12* with $T = 5$, *8* with $T = 10$, *2* with $T = 15$ and *5* with $T = 20$). In addition, we can see that MCKS becomes more difficult when increasing the number of divisions *T*. In fact, the number of times that CPLEX terminates with exceeding the capacity of RAM or exceeding the time limit increases from *18* with $T = 5$ to *25* with $T = 20$.

III.5 Conclusion

In this chapter, we consider the multiple choice knapsack problem with setup (MCKS). This problem can be used to model a wide range of concrete industrial problems, including order acceptance and production scheduling. We proposed a new cooperative approach that combines VNS and IP for the MCKS. Our matheuristic approach denoted VNS&IP is tested on a wide set of instances that are generated for MCKS. The results showed that CPLEX was able to optimally solve only 22.5% of these problems; the rest had unknown optimal values. The experimental results showed that VNS &IP produced good quality (optimal and near-optimal solutions) solutions in a short amount of time and allowed for the enhancement of the solution provided by CPLEX in 65 instances. Considering the promising performance of the

VNS&IP method presented in this work, further studies, some of which are currently underway in our laboratory, are needed to further extend the use of the space reduction technique to other general and critical problems.

Chapter IV

Cooperative approach for the Generalized Quadratic Multiple Knapsack Problem

IV.1 Introduction

In this Chapter, we address the 0-1 generalized quadratic multiple Knapsack problem (GQMKP). We use a linearization technique of the existing mathematical model and we propose a new cooperative approach that we called Matheuristic Variable Neighborhood Search (MVNS) combining variable neighborhood search with integer programing (IP) to solve the large sized instances. The matheuristic considers a local search technique with an adaptive perturbation mechanism to assign the classes to different knapsacks, and then once the assignment is identifed, applies the IP to select the items to allocate to each knapsack

The 0-1 generalized quadratic multiple knapsack problem (GQMKP) is NP-hard problem, since it is a generalization of the standard knapsack problem (KP) [20]. The GQMKP is reduced to KP when considering one knapsack, one class, no setup variables and a linear objective function. The GQMKP is described as a quadratic multiple knapsack problem (QMKP) with additional setup variables and knapsack-items preferences. The quadratic knapsack problem (QKP) is a particular case of QMKP, when considering only one knapsack. Practical applications of the GQMKP may be seen in production scheduling problems with setups and machines-products preferences. Case studies of GQMKP are provided in [9, 94].

Several variants of KP have been tackled in the literature [89]. Visée et al. [101] proposed a two-phased approach and branch and bound procedure to solve the bi-objective KP. Dudzinski and Walukiewicz [35] studied exact methods such as branch-and-bound and dynamic programming for several variants of KP. Johnson et al. [58] studied the graph version of the QKP and solved the linearized model with a branch-and-cut technique. Chaillou et al. [19] provided a branch and bound algorithm to solve the QKP. Billionet and Soutif [15] proposed a combination of a linear reformulation of the problem and a standard mixed integer programming tool to solve the QKP. Martello and Toth [76] discussed an upper bound using Lagrangian relaxation for multiple knapsack problem (MKP). Hiley and Julstrom [55] provided a greedy heuristic, a stochastic hill-climbing and a genetic algorithm to solve the QMKP. For the same problem, Sundar and Singh [98] developed an artificial bee colony algorithm, Garcia-Martinez et al. [43] provided an iterated greedy heuristic algorithm and Peng et al. [84] proposed an ejection chain method with an adaptive perturbation mechanism. The GQMKP is a generalization of the QMKP when considering three additional constraints: setup constraint, assignment conditions and knapsack preferences of the items. It has been presented by Sarac and Sipahioglu [94] who proposed a mathematical model, a genetic algorithm and a hybrid algorithm that combines genetic algorithm with a feasible value based modified sub gradient algorithm to solve the GQMKP. To solve the same problem, Chen and Hao [23] provided a memetic algorithm, where a backbone based crossover operator is integrated with a simulated annealing, and recently, Avci and Topaloglu [9] provided a multi-start iterated local search (MS-ILS) and made experiments on wide set of instances.

The hybridization technique between exact and metaheuristics approaches have been performed by many researchers during the last few decades. It provides interesting results as it takes advantages of both types of approaches [59]. A classification of algorithms combining local search techniques and exact methods is provided in [36]. The focus here is on the so called cooperative approaches using exact methods to strengthen local search techniques, and particularly on the matheuristics that combine metaheuristics and mathematical programming [53, 74]. Fernandes and Lourenco [39] applied a hybrid approach to solve different combinatorial optimization problems. Burke et al. [17] and Prandtstetter and Raidl [90] provided a combination of Integer Programming (IP) with Variable Neighborhood Search (VNS) for Highly-Constrained Nurse Roistering Problem, and car sequencing problem,

respectively. Lamghari et al. [71] proposed a combination of linear programming with variable neighborhood descent for scheduling production in open-pit mines. Vasquez and Hao [100] proposed a combination of linear programming with tabu search to solve the MKP problem. In this study, we combine IP with VNS to deal with GQMKP problem and make experimentation on the benchmark of Avci and Topaloglu [9] and Chen at al. [23].

The purpose of this work is to provide a solving approach for the GQMKP. We use a linearization technique of the existing mathematical model that, due to the complexity of the LGQMKP, cannot solve large test instances (see section IV.4). In fact, it is usually difficult to assign items to the whole sets of knapsacks. The GQMKP is a generalization of the knapsack problems when considering three additional constraints: setup constraint, assignment conditions and knapsack preferences of the items. In addition, the consideration of the knapsack-dependent cost related to each class of products and the knapsack-dependent profit associated to each item increases the complexity of the problem. Therefore, the design of a new approach providing high quality solutions in a reasonable computing time is quite challenging. This paper contains a matheuristic called matheuristic variable neighborhood search (MVNS) combining a VNS with an exact solving technique: local search techniques with an adaptive perturbation mechanism to include classes to knapsacks and integer programming (IP) to include items in each knapsack. Experimental results show the efficiency and the performance of the proposed matheuristic on a wide set of benchmark instances. Experimental results clearly show the competitiveness of the proposed approach compared to the best state-of-the-art solving techniques

The remainder of this paper is organized as following: Section IV.2 contains the mathematical formulation of the GQMKP. Section IV.3 contains our matheuristic approach combining VNS with IP. The experimental results and their interpretations are reported in Section IV.4 and, finally, the conclusions are outlined in Section IV.5.

IV.2 Mathematical model

We consider a set of K knapsacks each knapsack with a capacity c_k, $k \in \{1, \dots, K\}$, and a set of J items $j \in \{1, \dots, J\}$ which are classified into a set of R classes, $r \in \{1, \dots, R\}$. The main assumptions of the GQMKP are as follows:

- An item cannot be allocated to more than one knapsack.

- Items from the same class can be allocated to different knapsacks.

- For each class, the set of related items are allowed to be allocated to only predefined set of knapsacks.

- An item can be allocated to a knapsack only if its corresponding class is activated.

- A profit p_{jk} is considered while allocating item i to knapsack k

- A profit q_{ij} is considered if items i and j are allocated to the same knapsack.

- The activation of a class incurs a knapsack-independent setup time s_r

- If items belonging to the same class are allocated to the same knapsack, only one setup is needed for all.

The GQMKP problem consists of activating a set of classes in each knapsack, and determining the subset of items to be allocated from each class to each knapsack while maximizing the objective function without exceeding the capacity of each knapsack. Saraç and Sipahioglu [94] provided the following model for the GQMKP:

Sets:

BB_k : set of classes that can be activated in knapsack k

B_k : set of items that can be allocated to knapsack k

DD_r : set of knapsacks in which class r can be activated.

D_j : set of knapsacks to which item j can be allocated

Parameters:

p_{jk}: Profit obtained if item j is selected for knapsack k

q_{ij}: Profit obtained if items i and j are selected for the same knapsack

c_k: Working time capacity related to knapsack k,

w_j: Weight of item j (or processing time)

s_r: Setup time of the items that belong to class r

$t_{rj} = 1$ if item j in class r, 0 otherwise

$\eth_r$: The maximum number of knapsacks to which the items in class r can be assigned

U : A positive large number

Decision variables:

$x_{jk} = 1$ if item j is allocated to knapasack k ,0 otherwise

$y_{rk} = 1$ if any item in class r is allocated to knapasack k, 0 otherwise

Mathematical formulation:

$$\text{Max Z} = \sum_{\text{k}\epsilon\text{K}} \sum_{j\epsilon\, B_k} p_{jk} x_{jk} + \sum_{\text{k}\epsilon\text{K}} \sum_{i\epsilon\, B_k} \sum_{j\epsilon\, B_k (j>i)} q_{ij} x_{ik}\ x_{jk} \qquad (1)$$

Subject to

$$\sum_{j\epsilon\, B_k} w_j x_{jk} + \sum_{r\epsilon BB_k} s_r y_{rk} \leq c_k\ , \forall k \in K \qquad (2)$$

$$\sum_{\text{k}\epsilon D_j} x_{jk} \leq 1\ ;\ \forall j \in J \qquad (3)$$

$$\sum_{\text{j}\epsilon\, B_k} t_{rj} x_{jk} \leq U\ y_{rk}\ ;\ \forall k \in K, \forall r \in BB_k \qquad (4)$$

$$\sum_{\text{k}\epsilon\, DD_r} y_{rk} \leq \mathfrak{y}_r\ ;\ \forall r \in R \qquad (5)$$

$$x_{jk}, y_{rk} \in \{0,1\}\ ;\ \forall j \in J, \forall k \in K, \forall r \in R \qquad (6)$$

Equation (1) represents the objective function that is to maximize the total profit. Constraint (2) guarantees that the sum of the total weights of selected items and the class setup times consumption does not exceed knapsack capacity. Constraint (3) requires that each item can be allocated to only one knapsack. Constraint (4) guarantees that if any item in class r is selected for knapsack k, then the decision variable y_{rk} must be equal to 1. Constraint (5) ensures that the total number of knapsacks containing items belonging to class r cannot exceed the maximum number of knapsacks $\mathfrak{y}_r$. Finally, the constraint (6) ensures that the decision variables are binary.

Being inspired by the linearization technique provided in [16] for QKP problem, we provide the following linear model for the GQMKP, denoted LGQMKP:

$$\text{Max Z} = \sum_{\text{k}\epsilon\text{K}} \sum_{j\epsilon\, B_k} p_{jk} x_{jk} + \sum_{\text{k}\epsilon\text{K}} \sum_{i\epsilon\, B_k} \sum_{j\epsilon\, B_k (j>i)} q_{ij}\ z_{ijk} \qquad (7)$$

Constraints (2) to (6),

$$\boldsymbol{z_{ijk}} \leq x_{jk}\ ;\forall\ i,j\ \in J, \forall\ k\ \in K,\ i<j \qquad (8)$$

$$\boldsymbol{z_{ijk}} \leq x_{ik}\ ;\forall\ i,j\ \in\ J, \forall\ k\ \in K,\ i<j \qquad (9)$$

$$\boldsymbol{z_{ijk} \geq x_{jk} + x_{ik} - 1 \; ; \forall \; i,j \; \in \; J, \forall \; k \; \in K, \; i < j} \qquad (10)$$

$$z_{ijk} \in \{0,1\} \; ; \forall \; i,j \; \in \; J, \forall \; k \; \in K, \; i < j \qquad (11)$$

The linear expression 7 replaces the objective function 1. In addition to constraints 2-6, we provide the constraints 8-11 about the dummy binary variables z_{ijk}.

Using IP formulation (CPLEX 12.7) shows its limitation to solve LGQMKP due to the complexity of the problem. In fact, we show later in the experimental results (Section IV.4) that by using CPLEX 12.7, only 47 instances among 96 benchmark instances [9] are solved to optimality in less than 1 hour CPU time. For the rest of instances, the computation terminates with an out of memory or is stopped in 1 hour. Thus, we decided to invest in the development of a new cooperative approach can be seen as a matheuristic VNS combining VNS and IP. The keys of better performance for our approach instead aims to exploit the structure of LGQMKP, where the set of variables is partitioned into two levels, variables y_{rk} (first level variables) and variables x_{jk} (second level variables). Thus, we decided to invest in the development of a matheuristic VNS combining VNS and IP. The practical hardness of the problem comes from these two sets of variables that must be properly combined to reach an optimal solution. At the same time, once the classes are chosen, LGQMKP boils down to a several classical KP. Even if KP is known to be weakly NP-Hard, in practice it is well handled by nowadays ILP solvers [80]. We explain our new approach in the next section.

IV.3 Matheuristic VNS for GQMKP

From the VNS scheme, several other VNS approaches have been derived in [52]. In this paper we propose a new method combining VNS with integer programming for solving GQMKP. Within the approach, different mathematical programming formulations of sub problems are proposed and solved with exact solver. According to Hansen et al. [52] we call our variant of VNS as Matheuristic variable neighborhood search (MVNS). The main idea here is to partition the problem variables set into two levels: variables y_{rk} to be approximately defined using VNS and variables x_{jk} to be optimally defined using an ILP solved with CPLEX 12.7. In fact, once all y_{rk} variables are defined using VNS, the $LGQMKP$ could be seen as independent into K dependent knapsack problems $LGQMKP[Y_k]$. At a given knapsack k, $LGQMKP[Y_k]$ is a KP with a capacity $c_k - \theta_k$, where θ_k represents the

sum of capacity setup time of the classes activated in knapsack k and $R_{Y_k} = \{j \setminus t_{rj}=1, \forall r \in Y_k, \forall j \in B_k\}$ the set of items that can be allocated to knapsack k. The objective function is to maximize the total profit. The $LGQMKP[Y_k]$ can be formulated by a 0-1 KP linear program, using the allocation variables x_{jk} :

$$\text{Max } z_{opt}(k) = \sum_{j \in R_{Y_k}} p_{jk} x_{jk} + \sum_{i \in R_{Y_k}} \sum_{j \in R_{Y_k}(j>i)} q_{ij}\ z_{ijk} \qquad (12)$$

s.t.

$$\sum_{j \in B_k} w_j x_{jk} \leq c_k - \theta_k \quad ; \forall k \in K \qquad (13)$$

$$\boldsymbol{z_{ijk}} \leq x_{jk}\ ; \forall i,j \in R_{Y_k}, \forall k \in K,\ i < j \qquad (14)$$

$$\boldsymbol{z_{ijk}} \leq x_{ik}\ ; \forall i,j \in R_{Y_k}, \forall k \in K,\ i < j \qquad (15)$$

$$\boldsymbol{z_{ijk}} \geq x_{jk} + x_{ik} - 1\ ; \forall i,j \in R_{Y_k}, \forall k \in K,\ i < j \qquad (16)$$

$$x_{jk}, z_{ijk} \in \{0,1\}\ ; \forall i,j \in R_{Y_k}, \forall k \in K,\ i < j\ , \qquad (17)$$

where $\theta_k = \sum_{r \epsilon Y_k^1} s_r y_{rk}$

Each knapsack problem $LGQMKP[Y_k]$ is optimally solved to determine the best values of x_{jk}, which yield a feasible solution for $LGQMKP$ (or GQMKP). Let S_k be an optimal solution for $LGQMKP[Y_k]$ with the profit $z_{opt}(k)$. Thus, the proposed matheuristic MVNS provides the best combination of vector $Y = \bigcup_{k \in K} Y_k$ of a solution $S = \bigcup_{k \in K} S_k$ with a profit $Z = \bigcup_{k \in K} z_{opt}(k)$. Algorithm IV.1 shows the whole framework of our matheuristic VNS.

Algorithm IV.1: **MVNS**

Input: the set of neighborhood N_k (k=1,…, k_{max}), and the maximum number of iterations nb_iter

Initialize: Build an initial solution S_0 based on construction heuristic

S←S_0
stop←false;
While $(stop == false)$
  k←1;
  stop←true;
  Do
    S_p← **PERTURB&IP** (S) /* Shaking */
    S_1← $SWAP\&\ IP\ (S_p)$ /* local search SWAP with IP */
    S_2← $INSERT\&\ IP\ (S_1)$ /* local search INSERT with IP*/

If $(f(S_2) > f(S))$ **then**
S← S_2 ;
k←1;
stop←false;
else k← k+1**;**
While $(k == k_{max})$
EndWhile
Return S ;

The VNS is a method based on a systematic change of the neighborhood structures. It is provided by Maldenovic and Hansen [81] and has proven its efficiency on different scheduling problems: location routing [57], car sequencing problem [90], combinatorial optimization problems [34], etc. The VNS contains a shaking operator and local search operators that are developed based on the set of neighborhood structures. In our matheuristic MVNS, we consider shaking mechanism PERTURB&IP based on a perturbation move coupled with IP, and two local search techniques SWAP&IP and INSERT&IP, based on Swap and insert moves respectively, both coupled with IP. The MVNS matheuristic starts with an initial solution S. A set of neighborhoods N_k, $k = 1,\ldots,k_{max}$ is initialized. At each iteration the shaking mechanism PERTURB&IP is applied based on neighborhood N_k, then the two local search SWAP&IP and INSERT&IP are applied to obtain a new solution S_2. If this new solution is better than S, then this latter is updated and the process continues with the first neighborhood $N_1(S)$, otherwise the same steps are repeated with the next neighborhood N_{k+1}.

The construction heuristic, the shaking mechanism PERTURB&IP, and the two local search procedures SWAP&IP and INSERT&IP are detailed in the following subsections.

IV.3.1 Construction heuristic

To generate the initial solution for GQMKP, we propose a construction heuristic based on three successive phases:

- **First phase:** We transform the original problem $GQMKP$ to an equivalent formulation without q_{ij} profit, i.e. $q_{ij} = 0, for\ all\ i < j$. The new problem is denoted $GMKP$ and consists of maximizing the objective function (18), with constraints (2), (3), (4), (5) and (6).

$$\text{Max z} = \sum_{k \epsilon K} \sum_{j \epsilon B_k} p_{jk} x_{jk} \qquad (18)$$

- **Second phase:** we use the linear programming based heuristic provided in [96] to solve the linear relaxation of GMKP, denoted $RGMKP$, by relaxing the integrality constraints on variables x_{jk} (only the y_{rk} variables are binary). The fractional solution S_{LP} includes integer values y_{rk} and fractional and integer values x_{jk}. The reduced GMKP related to the fractional solution S_{LP} is referred to GMKP$[S_{LP}]$ that consists of fixing to 0 or 1 the fractional variables x_{jk}. The exact solution of the reduced problem GMKP$[S_{LP}]$ is a feasible solution and denoted $LB' = z$ (GMKP$[S_{LP}]$). We limit CPLEX computation time to 10 seconds to obtain an initial solution .

  - **Third phase :** the first feasible solution of GQMKP is $LB = LB' + \sum_{k \epsilon K} \sum_{i \epsilon B_k} \sum_{j \epsilon B_k (j>i)} q_{ij} x_{ik} \; x_{jk}$.

An illustration of the construction heuristic is provided by Algorithm IV.2.

Algorithm IV.2: Construction heuristic

Input : Instance of $GQMKP$
Output :A feasible solution LB of $GQMKP$
Step 1 :Transform the $GQMKP$ to $GMKP$ where, q_{ij} = 0 for all i < j
Step 2 :$S_{LP} \leftarrow$ Solve ($RGMKP$)
Step 3 :$LB' \leftarrow$ Solve (GMKP$[S_{LP}]$)
Step 4 :Return the resulting feasible solution LB of GQMKP
where $LB = LB' + \sum_{k \epsilon K} \sum_{i \epsilon B_k} \sum_{j \epsilon B_k (j>i)} q_{ij} x_{ik} \; x_{jk}$

IV.3.2 SWAP&IP

A Swap-based local search requires the definition of a neighborhood structure using simple moves so as to produce a set of neighbor solutions which permits to explore more search spaces and thus provides high quality solutions. The considered swap process consists of:

- Permuting two classes r_1 and r_2 activated in different knapsacks k_1 and k_2 i.e. $r_1 \in Y_{k_1}$ and $r_2 \in Y_{k_2}$. More precisely, we change the values of setup variables $y_{r_1 k_1}$ and $y_{r_2 k_2}$.

- Removing a class r_1 activated in knapsack k_1 and replacing it by a free class $r_2 \in Y_{free}$, where $Y_{free} = \{r \backslash y_{rk} = 0, \forall\, r \in \{1, \dots, R\}, \forall\, k \in \{1, \dots, K\}\}$. More precisely, we change the value of setup variables y_{rk} from 1 to 0 and vice versa.

After SWAP movements, the new sub-problems: $LGQMKP[Y_{k_1}]$ and $LGQMKP[Y_{k_2}]$ are solved to optimally using IP. If the new solution is better than the best feasible solution value i.e. $(z_{opt}(k_1) + z_{opt}(k_2)) > (Z(k_1) + Z(k_2))$, then it is considered as a new initial solution for a next SWAP process. We solve to optimality a $LGQMKP[Y_{k_1}]$ and $LGQMKP[Y_{k_2}]$, but indeed $Y = \bigcup_{k \in K} Y_k$ is not guaranteed to be optimal for $LGQMKP$. The SWAP&IP procedure continues until no improvement is obtained. Algorithm IV.3 details the *SWAP&IP* procedure.

Algorithm IV.3: SWAP&IP

Input: Instance data & best solution found LB
Output: A feasible solution S

```
S← LB
do
  Improve ← 0;
  For k1 ← 1 to K do
      n ← card(Yk1); /* Number of classes in knapsack k1 */
      For x ← 1 to n do
          r1 ← Yk1[x]; /* x^th selected class in knapsack k1 */
          For k2 ← k1 + 1 to K + 1 do /* YK+1 contain the free classes */
              nb ← card(Yk2); /* Number of classes in knapsack k2 */
              For l ← 1 to nb do /*Swap class r1 by each class r2 */
                  r2 ← Yk2[l] ; /* l^th selected class in knapsack k2 */
                  yr1k1 ← 0;
                  yr2k2 ← 0;
                  yr1k2 ← 1;
                  yr2k1 ← 1;
                  zopt(k1)← solve LGQMKP[Yk1] /* using IP */
                  zopt(k2)← solve LGQMKP[Yk2]
                  LB←Z − (Z(k1) + Z(k2)) + (zopt(k1) + zopt(k2))
                  If (LB > S) then
                      Improve ← 1;
                      S ← LB
                  EndIf
              EndFor
          EndFor
      EndFor
  EndFor
```

EndFor
While (improve==1)
Return S

IV.3.3 INSERT&IP

The Insert-based local search is based on a neighborhood search which generates a new solution by removing the class r_1 from knapsack k_1 (change the value of the setup variable $y_{rk_1} \in Y_{k_1}$ from 1 to 0) and then inserting it into another knapsack k_2. (Change the value of the setup variable y_{rk_2} from 0 to 1). So we search for another possible vector Y_K within the sub-problem by inserting each setup variables y_{rk} in different knapsacks. The IP is applied to optimally solve the $LGQMKP[Y_{k_1}]$ and $LGQMKP[Y_{k_2}]$, but indeed Y is not guaranteed to be optimal for GQMKP. So we search for another possible combination of y_{rk} by progressively inserting each class $r \in R$ in different knapsacks and applying IP to optimally fix x_{jk} variables, while keeping only a subset of potentially good nodes as candidates for further exploration. The INSERT&IP algorithm starts with the best solution LB returned by SWAP&IP, and then proceeds by choosing the best neighbor solutions to LB. The procedure is terminated once no improvement is obtained. Algorithm IV.4 details the *INSERT&IP* procedure.

Algorithm IV.4: *INSERT&IP*

Input: Instance data & best solution LB found by SWAP&IP

Output: A best feasible solution S

do
$Improve \leftarrow 0$;
For $r_1 \leftarrow 1$ *to* R **do**
$kp \leftarrow \{k \backslash y_{r_1 k} = 1, \forall\, k \in 1, \dots, K\}$; /* *set of knapsack contain class r*/
$n \leftarrow card(kp)$; /* Number of knapsack that contain class r */
For $x \leftarrow 1$ *to* n **do**
$k_1 \leftarrow kp[x]$;
For $k_2 \leftarrow 1$ *to* K **do**

If $(y_{r_1k_2} = 0)$ **then**
$y_{r_1k_1} \leftarrow 0$; $y_{r_1k_2} \leftarrow 1$; /* Insert r_1 in k_2 and delete it from k_1 */
$z_{opt}(k_1) \leftarrow solve\ LGQMKP[Y_{k_1}]$ /* using IP */

$z_{opt}(k_2) \leftarrow solve\ LGQMKP[Y_{k_2}]$

$LB \leftarrow Z - (Z(k_1) + Z(k_2)) + (z_{opt}(k_1) + z_{opt}(k_2))$

If $(LB > S)$ **then**
$S \leftarrow LB$; $Improve \leftarrow 1$;
EndIf
EndIf
EndFor
EndFor
EndFor
While (improve ==1)
Return S

IV.3.4 PERTURB&IP

The design of the perturbation mechanism PERTURB&IP is crucial for the performance of the MVNS algorithm. If the mechanism provides too small perturbation, local search may return to the previously visited local optimum and no further improvement is obtained i.e. quick convergence to a local optimum. PERTURB&IP consists of strongly perturbing a part of the current solution to jump the local optima and obtain a new starting solution. Algorithm IV.5 provides a description of PERTURB&IP procedure.

Algorithm IV.5: *Perturb&IP*

Input: Instance data & best solution found LB
Output: best solution S
S←LB
$p \leftarrow 1$;
N_{min}←Solve GQMKP$_{min}$
N_{max}←Solve GQMKP$_{max}$
For all N in [$\lceil N_{min} \rceil$, $\lfloor N_{max} \rfloor$ **] do**
Select a random set of N classes , from R
Do
Y←**Randomly** assigned the y_{rk}variables
LB←Apply IP to **optimally** solve $LGQMKP[Y]$
If $(f(LB) > \mathfrak{y}\ f(S))$ **then**

S←LB;

$p \leftarrow 1$;

Else

p←p+1;

End if

while $p \leq p_{max}$

End for

return S

Two phases were applied iteratively in order to simulate this jumping principle:

- **Phase 1.** Let N the number of activated classes leading to an optimal solution of LGQMKP (or GQMKP). N is bounded straight forwardly by solving two linear continuous problems: minimize and maximize $\sum_{k\in K}\sum_{r\in BB_k} y_{rk}$ subject to constraints (2-5) , (8-11) and to an additional constraint ensuring that the total profit must be strictly greater than the best solution value LB (19) and the non-integrality of variables x_{jk} and y_{rk} (20):

$$\sum_{k\epsilon K}\sum_{j\epsilon\, B_k} p_{jk}x_{jk} + \sum_{k\epsilon K}\sum_{i\epsilon\, B_k}\sum_{j\epsilon\, B_k(j>i)} q_{ij}\, z_{ijk} \geq LB + 1 \quad (19)$$

$$0 \leq x_{jk}, y_{rk} \leq 1 \quad ,\forall j \in J, \forall k \in K, \forall r \in R \quad (20)$$

By solving the corresponding ILP formulations, denoted GQMKP_{min} and GQMKP_{max}, we obtain the minimum and maximum numbers of classes N_{min} and N_{max}. The first step is to randomly select N classes $N \in [N_{min}, N_{max}]$ and randomly assign activate the selected N classes in different knapsacks i.e. randomly fixing y_{rk}variables.

- **Phase 2.** The second phase consists of optimally solve the $LGQMKP[Y]$ using IP i.e. fixing x_{jk} variables. The resulting solution S is accepted if (f(S) > ŋ f(LB)), where ŋ that is a constant value between 0 and 1. PERTURB&IP terminates when the total number of applied moves (perturbations) reaches p_{max}.
-

IV.4 Computational results

For computation, our approach is implemented and run using C language and CPLEX 12.7 solver on a personal computer with 2.4 GHZ intel core 2 duo B960 processor and 4 GB of memory. In order to test the performance of the MVNS for the GQMKP, two sets of benchmark instances [9, 23] are considered:

- **First Set**: This set is composed of 48 small-sized instances which are characterized by their number of items J =30, number of knapsacks k ∈ {1, 3}, number of classes r ∈ {3, 15}, density (the percentage of those values for the p_j and q_{ij} profit parameters different from zero) d ∈ {0.25, 1.00}.
- **Second Set:** Includes 48 large-sized instances with the number of items J = 300, number of knapsacks k ∈ {10, 30}, number of classes r ∈ {30, 150}, density d ∈ {0.25, 1.00}.

All data sets are available at https://goo.gl/dv3tfA. Based on these data sets, we made a comparison between our LGQMKP model (solved with CPLEX 12.7), our MVNS, the MA [23] and the MS-ILS [9] that are, to the best of our knowledge, the best algorithms provided in literature to deal with GQMKP. We note that tests in [9] were carried on an Intel core 2 duo T7500 CPU @ 2.2 GHZ, and tests in [23] were carried on an AMD Opteron 4184 processor 2.8GHz and 2GB RAM.

The parameters of our approach MVNS are set so as to get a satisfactory trade-off between quality solution and running time: the maximum number of consecutive failed iterations k_{max} is fixed to R. The perturbation length p_{max} is fixed to K. The parameter ϑ is fixed to 0.8 to relax the acceptance condition.

Before the experimentation (Section IV.4.2), we provide a performance analysis of the MVNS components (Section IV.4.1).

IV.4.1 Performance analysis of the MVNS components

We study here the performance of the main components of our matheuristics, mainly the construction heuristic and the combination of the two local search techniques *SWAP&IP*, *INSERT&IP* and the perturbation mechanism *PERTURB&IP*. The results graphically displayed in figure IV.1 illustrate a comparison in terms of quality solution, where the vertical

axis shows the gap between the MVNS component solution and the MS-ILS solution:: $\text{Gap}(\%) = 100 * \left(\frac{\text{MS-ILS}_{\text{sol}} - \text{MVNS } Component_{\text{sol}}}{\text{MS-ILS}_{\text{sol}}}\right)$. We consider fir this study, five large instances denoted Ins 1-1, Ins 2-2, Ins 11-3, Ins 12-2 and Ins 19-3 (a selection of instances from experimentation table IV.2).

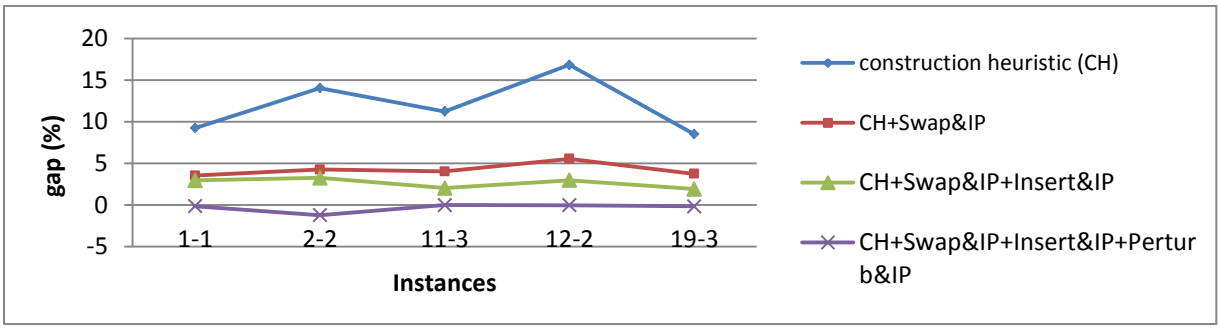

Figure IV.1: Effect of MVNS components

Figure IV.1 shows that the constructive heuristic plays an important role in the overall performance of the provided MVNS approach: The initial solutions are close to the solutions provided by MS-ILS with an average gap lower than 12%. We consider the application of our matheuristic with one local search technique (*SWAP&IP*), two local search techniques (*SWAP&IP* and *INSERT&IP*), and three local search techniques (*SWAP&IP*, *INSERT&IP* and *PERTURB&IP*). Figure IV.1 shows a comparison between these three combinations in terms of average Gap (%) with the MS-ILS. By adding *SWAP&IP*, we observe a higher improvement, for all the five large instances, compared to using only Construction heuristic. However, by adding *Insert&IP*, we observe a slight advantage. The perturbation mechanism Perturb &IP enables MVNS to produce better solutions. More precisely, for the five large instances (300 items), the gap on average is 4,22% when applying only the SWAP&IP, 2.63% when applying SWAP&IP and INSERT&IP and -0.3% by adding the perturbation mechanism PERTURB&IP.

IV.4.2 Experimentation

Tables IV.1 and IV.2 summarize the results of the LGQMKP model (solved with CPLEX 12.7), MA, MS-ILS and MVNS on GQMKP instances. In each of these tables, the first two columns present the number of knapsacks K and the number of classes R. We note that the column *Best known* reports the best value reported by any of the compared

algorithms (MA, MS-ILS, MVNS). The columns dev report the standard deviation from the best known: $\text{dev}(\%) = 100 * \left(\frac{\text{Best known} - \text{algorithm}_{\text{sol}}}{\text{Best known}}\right)$. The next four columns show the corresponding results provided by CPLEX (objective value OBj, computation time CPU and deviation dev), MA, MS-ILS and MVNS (best value $best$, average value $Avg.sol$ average computation time CPU and deviation dev). We note that CPLEX is stopped at a limit of computation time equals to 1 hour. For the proposed MVNS matheuristic, we report the best and average solutions of 30 independent runs on each benchmark instance. Finally, the last column presents the best known. The time unit in this table for the CPU is in seconds. The detailed results are available on the following link: https://goo.gl/fbFBgV.

Table IV.1 presents the computational results obtained for the first set of benchmark instances. The results show that CPLEX for LGQMKP is effective and finds the optimal solutions for 47 among 48 instances (all instances except 23-3 instance). Our approach MVNS provides a solution equal to CPLEX for these 47 instances and reaches the best known solution for instance 23-3. MVNS and MS-ILS provide the best know solutions for all first set instances while MA obtain 45 best known solutions among 48 instances with an insignificant average dev equal to 0,44%. When analyzing the average solutions, we observe that MVNS has produced the same results for all instances excepts 32-1 instance. Based on the comparison of the average results of MVNS and MS-ILS, we see that MVNS outperforms MS-ILS and MA on six instances and twelve instances, respectively. This result indicates the robustness of the matheuristic approach.

Table IV.1. Computational results obtained from the first set of benchmark instances

Ins.	K	R	$LGQMKP$			MA				$MS-ILS$				$MVNS$				$Best\ Known$
			Obj	CPU	Dev (%)	$best$	$Avg.sol$	CPU	Dev (%)	$best$	$Avg.sol$	CPU	Dev (%)	$best$	$avg.sol$	CPU	Dev (%)	
5-1	3	15	**2835.30**	7.12	0.00	**2835.30**	2828.22	1.23	0.00	**2835.30**	2835.30	3.24	0.00	**2835.30**	2835.30	2.23	0.00	2835.30
5-2			**3304.80**	153.92	0.00	3293.48	3293.90	0.83	0.34	**3304.80**	3293.48	1.65	0.00	**3304.80**	3304.80	2.04	0.00	3304.80
5-3			**1678.00**	5.08	0.00	**1678.00**	1678.00	0.01	0.00	**1678.00**	1678.00	3.17	0.00	**1678.00**	1678.00	1.86	0.00	1678.00
6-1	1	3	**346.40**	0.68	0.00	**346.40**	346.40	0.01	0.00	**346.40**	346.40	2.48	0.00	**346.40**	346.40	2.25	0.00	346.40
6-2			**554.00**	0.11	0.00	**554.00**	554.00	0.01	0.00	**554.00**	554.00	1.41	0.00	**554.00**	554.00	2.09	0.00	554.00
6-3			**428.70**	0.20	0.00	**428.70**	428.70	0.01	0.00	**428.70**	428.70	1.75	0.00	**428.7**	428.70	1.16	0.00	428.70
8-1	3	15	**309.21**	7.38	0.00	**309.21**	309.21	0.91	0.00	**309.21**	309.21	2.25	0.00	**309.21**	309.21	2.02	0.00	309.21
8-2			**353.85**	2.98	0.00	**353.85**	353.69	0.11	0.00	**353.85**	353.85	2.97	0.00	**353.85**	353.85	2.09	0.00	353.85
8-3			**541.57**	2.56	0.00	**541.57**	541.57	0.03	0.00	**541.57**	541.57	2.85	0.00	**541.57**	541.57	2.25	0.00	541.57
15-1	1	3	**91.54**	1.39	0.00	**91.54**	91.54	0.32	0.00	**91.54**	91.54	1.60	0.00	**91.54**	91.54	2.22	0.00	91.54
15-2			**306.38**	0.56	0.00	**306.38**	306.38	0.02	0.00	**306.38**	306.38	2.85	0.00	**306.38**	306.38	2.13	0.00	306.38
15-3			**75.62**	1.50	0.00	**75.62**	75.45	0.37	0.00	**75.62**	75.62	2.77	0.00	**75.62**	75.62	1.38	0.00	75.62
18-1	1	3	**5387.70**	4.09	0.00	**5387.70**	5387.70	0.01	0.00	**5387.70**	5387.70	2.11	0.00	**5387.70**	5387.70	2.03	0.00	5387.70
18-2			**8551.08**	0.33	0.00	**8551.08**	8551.08	0.00	0.00	**8551.08**	8551.08	3.03	0.00	**8551.08**	8551.08	1.29	0.00	8551.08
18-3			**7760.51**	0.70	0.00	**7760.51**	7760.51	0.00	0.00	**7760.51**	7760.51	1.43	0.00	**7760.51**	7760.51	1.05	0.00	7760.51
20-1	1	15	**1599.85**	0.44	0.00	**1599.85**	1599.85	0.01	0.00	**1599.85**	1599.85	1.99	0.00	**1599.85**	1599.85	1.02	0.00	1599.85

20-2			**925.59**	1.73	0.00	**925.59**	925.59	0.01	0.00	**925.59**	925.59	2.81	0.00	**925.59**	925.59	1.55	0.00	925.59
20-3			**931.33**	0.63	0.00	**931.33**	931.33	0.01	0.00	**931.33**	931.33	2.83	0.00	**931.33**	931.33	1.42	0.00	931.33
22-1	3	3	**1923.61**	1.80	0.00	1904.86	1904.86	0.02	0.98	**1923.61**	1911.11	3.30	0.00	**1923.61**	1923.61	2.03	0.00	1923.61
22-2			**1314.09**	0.16	0.00	**1314.09**	1314.09	0.01	0.00	**1314.09**	1314.09	1.33	0.00	**1314.09**	1314.09	2.15	0.00	1314.09
22-3			**1799.09**	0.46	0.00	**1799.09**	1799.09	0.02	0.00	**1799.09**	1799.09	2.04	0.00	**1799.09**	1799.09	1.13	0.00	1799.09
23-1	3	3	**471.00**	3600.16	0.00	**471.00**	471.00	0.02	0.00	**471.00**	471.00	2.53	0.00	**471.00**	471.00	2.05	0.00	471.00
23-2			**959.70**	292.99	0.00	**959.70**	959.70	0.06	0.00	**959.70**	959.70	1.02	0.00	**959.70**	959.70	2.99	0.00	959.70
23-3			1233.00	1791.67	0.64	**1241.00**	1241.00	0.32	0.00	**1241.00**	1241.00	1.20	0.00	**1241.00**	1241.00	2.81	0.00	1241.00

Table IV.1. Computational results obtained from the first set of benchmark instances (cont)

inst	K	R	$LGQMKP$			MA				$MS-ILS$				$MVNS$				$Best\ known$
			Obj	CPU	dev	$best$	$Avg.sol$	CPU	dev	$best$	$Avg.sol$	CPU	dev	$best$	$avg.sol$	CPU	dev	
25-1	3	15	**2118.33**	5.79	0.00	**2118.33**	2118.33	1.52	0.00	**2118.33**	2118.33	1.34	0.00	**2118.33**	2118.33	1.05	0.00	2118.33
25-2			**4262.64**	6.38	0.00	**4262.64**	4195.05	1.66	0.00	**4262.64**	4193.01	1.12	0.00	**4262.64**	4262.64	0.98	0.00	4262.64
25-3			**2962.06**	7.43	0.00	**2962.06**	2962.06	1.03	0.00	**2962.06**	2962.06	1.02	0.00	**2962.06**	2962.06	1.25	0.00	2962.06
26-1	1	15	**1747.60**	10.18	0.00	**1747.60**	1747.60	0.01	0.00	**1747.60**	1747.60	2.66	0.00	**1747.60**	1747.60	0.93	0.00	1747.60
26-2			**2433.60**	2.76	0.00	**2433.60**	2433.60	0.01	0.00	**2433.60**	2433.60	1.20	0.00	**2433.60**	2433.60	1.17	0.00	2433.60
26-3			**2293.20**	1.14	0.00	**2293.20**	2293.20	0.01	0.00	**2293.20**	2293.20	1.32	0.00	**2293.20**	2293.20	2.07	0.00	2293.20
27-1	1	15	**2247.95**	0.51	0.00	**2247.95**	2247.95	0.01	0.00	**2247.95**	2247.95	2.76	0.00	**2247.95**	2247.95	1.99	0.00	2247.95

27-2			**1966.52**	0.11	0.00	**1966.52**	1966.52	0.01	0.00	**1966.52**	1966.52	1.05	0.00	**1966.52**	1966.52	2.14	0.00	1966.52
27-3			**1383.49**	0.13	0.00	**1383.49**	1383.49	0.01	0.00	**1383.49**	1383.49	1.09	0.00	**1383.49**	1383.49	1.86	0.00	1383.49
28-1			**978.80**	0.64	0.00	**978.80**	978.07	0.12	0.00	**978.80**	978.80	2.67	0.00	**978.80**	978.80	1.23	0.00	978.80
28-2	1	15	**4036.00**	0.19	0.00	**4036.00**	4035.62	0.04	0.00	**4036.00**	4036.00	1.12	0.00	**4036.00**	4036.00	1.11	0.00	4036.00
28-3			**2634.00**	0.18	0.00	**2634.00**	2634.00	0.01	0.00	**2634.00**	2634.00	1.14	0.00	**2634.00**	2634.00	1.19	0.00	2634.00
29-1			**1935.80**	9.89	0.00	1567.60	1520.33	0.19	19.02	**1935.80**	1935.80	2.46	0.00	**1935.80**	1935.80	2.01	0.00	1935.80
29-2	3	3	**2820.00**	24.52	0.00	2782.00	2782.00	0.10	1.35	**2820.00**	2820.00	1.08	0.00	**2820.00**	2820.00	1.64	0.00	2820.00
29-3			**3285.60**	11.81	0.00	**3285.60**	3285.60	0.05	0.00	**3285.60**	3285.60	1.03	0.00	**3285.60**	3285.60	1.45	0.00	3285.60
30-1			**721.39**	17.49	0.00	**721.39**	717.27	0.40	0.00	**721.39**	719.58	2.47	0.00	**721.39**	721.39	2.01	0.00	721.39
30-2	3	3	**612.59**	7.91	0.00	**612.59**	612.59	0.03	0.00	**612.59**	612.59	1.02	0.00	**612.59**	612.59	1.89	0.00	612.59
30-3			**1032.35**	3.17	0.00	**1032.35**	1032.35	0.04	0.00	**1032.35**	1031.94	1.88	0.00	**1032.35**	1032.35	1.55	0.00	1032.35
31-1			**491.90**	166.75	0.00	**491.90**	491.90	1.52	0.00	**491.90**	491.90	1.98	0.00	**491.90**	491.90	3.11	0.00	491.90
31-2	3	15	**640.00**	113.21	0.00	**640.00**	640.00	0.49	0.00	**640.00**	640.00	1.21	0.00	**640.00**	640.00	2.05	0.00	640.00
31-3			**526.10**	808.19	0.00	**526.10**	526.10	5.37	0.00	**526.10**	526.10	1.16	0.00	**526.10**	526.10	2.04	0.00	526.10
32-1			**11425.20**	1.63	0.00	**11425.20**	11271.90	0.02	0.00	**11425.20**	11283.21	2.61	0.00	**11425.20**	11393.75	2.53	0.00	11425.20
32-2	1	3	**15914.20**	0.47	0.00	**15914.20**	15914.20	0.00	0.00	**15914.20**	15914.20	1.03	0.00	**15914.20**	15914.20	1.13	0.00	15914.20
32-3			**19273.50**	0.02	0.00	**19273.50**	19273.50	0.00	0.00	**19273.50**	19273.50	1.09	0.00	**19273.50**	19273.50	1.25	0.00	19273.50
AVG			2 738.02	147.48	0.01	2 729.09	2 723.25	0.35	0.45	2 738.18	2 733.23	1.92	0.00	2 738.18	2 735.22	1.77	0.00	2 738.18

Table IV.2. Computational results obtained from the second set of benchmark instances

Ins.	K	R	$LGQMKP$			MA				$MS-ILS$				$MVNS$				$Best\ Known$
			Obj	CPU	dev	$best$	$Avg.sol$	CPU	dev	$best$	$Avg.sol$	CPU	dev	$best$	$avg.sol$	CPU	dev	
1-1	10	30	*	3603.87	*	5093.06	5074.50	7419.16	0.29	5100.54	5016.82	3793.42	0.14	**5107.80**	5102.17	225.23	0.00	5107.80
1-2			*	56.24	*	4848.58	4830.20	8101.91	0.84	4858.84	4784.07	4493.57	0.63	**4889.58**	4889.58	295.09	0.00	4889.58
1-3			*	56.27	*	5896.01	5876.05	6823.41	0.12	**5902.86**	5823.73	4710.79	0.00	**5902.86**	5898.23	259.01	0.00	5902.86
2-1	30	150	*	98.62	*	2607.84	2601.31	3530.13	0.01	**2608.12**	2557.22	5175.50	0.00	**2608.12**	2601.20	443.04	0.00	2608.12
2-2			*	98.87	*	**2285.32**	2281.63	3570.48	0.00	2257.88	2249.62	3925.30	1.20	**2285.32**	2285.32	525.11	0.00	2285.32
2-3			*	111.68	*	2578.14	2573.40	2946.75	0.1	**2580.62**	2574.96	3464.23	0.00	**2580.62**	2577.08	390.03	0.00	2580.62
3-1	10	150	16760.30*	3610.46	*	32189.10	32147.30	2693.57	0.08	32210.80	32163.74	1734.37	0.02	**32216.20**	32210.32	307.98	0.00	32216.20
3-2			4640.30*	3601.01	*	40302.40	40169.70	1437.15	0.13	**40354.90**	40239.63	2399.47	0.00	**40354.90**	40354.90	523.02	0.00	40354.90
3-3			3196.10*	3628.88	*	32766.70	32749.40	3414.05	0.02	32768.20	32704.85	4220.01	0.01	**32772.40**	32772.40	309.15	0.00	32772.40
4-1	10	150	*	58.26	*	9045.80	9027.86	4323.70	0.03	**9048.40**	9029.01	2720.37	0.00	**9048.40**	9048.40	223.06	0.00	9048.40
4-2			*	60.70	*	8465.00	8448.00	4871.10	0.04	**8468.50**	8425.58	2207.61	0.00	**8468.50**	8459.36	255.03	0.00	8468.50
4-3			*	60.66	*	8491.30	8475.10	4467.05	0.07	8494.20	8450.93	2058.68	0.04	**8497.20**	8490.23	189.45	0.00	8497.20
7-1	10	30	*	61.74	*	68129.00	68029.40	3314.59	0.05	**68165.50**	68060.77	1669.41	0.00	**68165.50**	68160.81	301.09	0.00	68165.50
7-2			*	59.71	*	65616.80	65546.20	2542.84	0.04	**65643.50**	65559.26	1943.83	0.00	**65643.50**	65643.50	208.01	0.00	65643.50
7-3			*	68.10	*	69397.60	69279.30	3104.20	0.06	**69440.90**	69295.39	2304.84	0.00	**69440.90**	69440.90	205.21	0.00	69440.90
9-1	30	30	*	94.75	*	9252.47	9242.60	1485.96	0.04	**9256.47**	9245.74	3262.62	0.00	**9256.47**	9256.47	558.82	0.00	9256.47

9-2			*	98.61	*	13007.30	12988.90	3120.53	0.05	13009.08	12943.85	3215.71	0.03	**13013.20**	13013.20	398.86	0.00	13013.20
9-3			*	97.84	*	16372.00	16359.20	2822.24	0.09	**16385.97**	16364.29	3547.43	0.00	**16385.97**	16385.97	480.2	0.00	16385.97
10-1	30	30	*	41.04	*	13196.30	13125.80	3761.90	0.14	**13214.66**	11147.24	5078.34	0.00	**13214.66**	13214.66	508.11	0.00	**13214.66**
10-2			*	511.88	*	13003.30	12779.20	3796.61	0.09	**13015.08**	11209.88	5482.66	0.00	**13015.08**	13015.08	514.2	0.00	**13015.08**
10-3			*	3604.61	*	13057.00	13008.60	4114.58	0.09	**13068.47**	11672.09	4046.23	0.00	**13068.47**	13068.47	397.03	0.00	**13068.47**
11-1	10	30	534.00*	3600.86	*	7116.50	7103.55	711.63	0.08	**7121.90**	7108.17	2536.74	0.00	**7121.90**	7121.90	306.05	0.00	**7121.90**
11-2			339.20*	3602.37	*	6771.50	6758.19	537.44	0.05	**6774.70**	6760.15	2546.20	0.00	**6774.70**	6774.70	414.89	0.00	**6774.70**
11-3			539.60*	1975.36	*	7745.10	7726.96	911.53	0.03	**7747.10**	7705.36	2952.58	0.00	**7747.10**	7747.10	262.4	0.00	**7747.10**

Table IV.2. Computational results obtained from the second set of benchmark instances (cont)

Ins.	K	R	$LGQMKP$			MA				$MS-ILS$				$MVNS$				$Best\ Known$
			Obj	CPU	dev	$best$	$Avg.sol$	CPU	dev	$best$	$Avg.sol$	CPU	dev	$best$	$avg.sol$	CPU	dev	
12-1	30	150	*	103.53	*	592	59137.7	6140.09	0.60	**59592.00**	59381.42	4665.91	0.00	**59592.00**	59592	301.17	0.00	**59592.00**
12-2			*	90.25	*	614	61181.7	4800.17	0.40	61725.20	61449.70	4163.09	0.02	**61737.40**	61730	360.23	0.00	**61737.40**
12-3			*	92.71	*	608	60749.7	5156.66	0.44	**61165.70**	60988.89	4541.59	0.00	**61165.70**	61165	489.20	0.00	**61165.70**
13-1	30	150	*	724.18	*	**421**	4194.59	1901.71	0.00	4196.20	4132.31	1176.32	0.33	4207.20	4198.	190.13	0.07	**4210.10**
13-2			*	584.78	*	**413**	4136.01	1318.51	0.00	4119.60	4086.26	1181.03	0.49	**4139.90**	4139.	201.25	0.00	**4139.90**
13-3			*	581.17	*	473	4717.27	2123.22	0.22	4722.10	4666.32	1160.96	0.49	**4745.10**	4742.	186.17	0.00	**4745.10**
14-1	10	30	26750.32*	3601.66	*	**268**	26868.6	4.61	0.00	**26868.60**	26868.60	1416.34	0.00	**26868.60**	26868	213.07	0.00	**26868.60**

14-2			3143.93*	76.06	*	259	25720.0	304.66	0.18	25885.67	25743.61	1290.07	0.35	**25976.20**	25972	287.06	0.00	**25976.20**
14-3			1825.67*	3601.51	*	**314**	31448.2	301.21	0.00	**31448.20**	31444.55	1353.25	0.00	**31448.20**	31448	266.52	0.00	**31448.20**
16-1	30	30	7622.90*	600.21	*	141	14060.9	1932.42	0.27	**14166.80**	14086.34	1011.32	0.00	**14166.80**	14166	198.03	0.00	**14166.80**
16-2			*	1017.00	*	166	16577.6	1634.65	0.00	**16612.40**	16582.07	811.82	0.00	**16612.40**	16612	183.06	0.00	**16612.40**
16-3			*	840.83	*	142	14210.1	2391.58	0.07	**14251.00**	14230.64	940.71	0.00	**14251.00**	14251	192.71	0.00	14251.00
17-1	30	30	*	106.86	*	**415**	4147.09	1221.69	0.00	**4157.20**	4149.12	3242.94	0.00	**4157.20**	4157.	198.65	0.00	4157.20
17-2			*	101.97	*	390	3891.48	1500.73	0.25	3892.00	3881.10	3009.30	0.49	**3911.00**	3911	103.33	0.00	3911.00
17-3			*	93.66	*	**376**	3767.67	1444.65	0.00	3756.80	3744.68	3460.16	0.29	**3767.70**	3767.	196.13	0.00	3767.70
19-1	10	150	930.47*	3600.99	*	686	6866.33	843.08	0.05	**6873.07**	6853.02	1641.79	0.00	**6873.07**	6865.	153.69	0.00	6873.07
19-2			691.96*	3601.31	*	802	7831.85	1847.38	0.18	**8042.79**	7888.13	2137.21	0.00	**8042.79**	8042.	201.06	0.00	8042.79
19-3			554.83*	3601.83	*	**815**	8154.87	1410.02	0.00	8142.84	8131.62	1724.09	0.15	**8155.05**	8155.	113.20	0.00	8155.05
21-1	10	150	*	59.09	*	222	22187.0	7570.7	0.04	22210.23	22121.54	3100.56	0.09	**22230.23**	22225	167.23	0.00	22230.23
21-2			*	64.96	*	252	25199.7	5544.17	0.05	**25266.50**	25165.91	3867.18	0.00	**25266.50**	25260	200.03	0.00	25266.50
21-3			*	80.21	*	245	24541.2	8984.81	14.06	28565.63	27847.15	3914.01	0.1	**28593.40**	28589	188.14	0.00	28593.40
24-1	30	150	*	53.73	*	526	52253.3	91.59	1.25	53318.76	53173.81	3103.33	0.00	**53320.50**	53312	203.48	0.00	53320.50
24-2			*	8.37	*	577	57513.4	2868.01	3.27	59712.09	59169.73	3190.82	0.02	**59723.20**	59719	203.60	0.00	59723.20
24-3			*	5.58	*	526	52361.6	77.34	0.81	**53073.72**	52929.65	3456.26	0.00	**53073.72**	53073	290.73	0.00	53073.72
AVG			*	1086.56	*	218 99.3	21831.6 7	3025.75	0.51	22067.96	21871.01	2896.87	0.10	22075.09	22072 90	287.25	0.00	22075.15

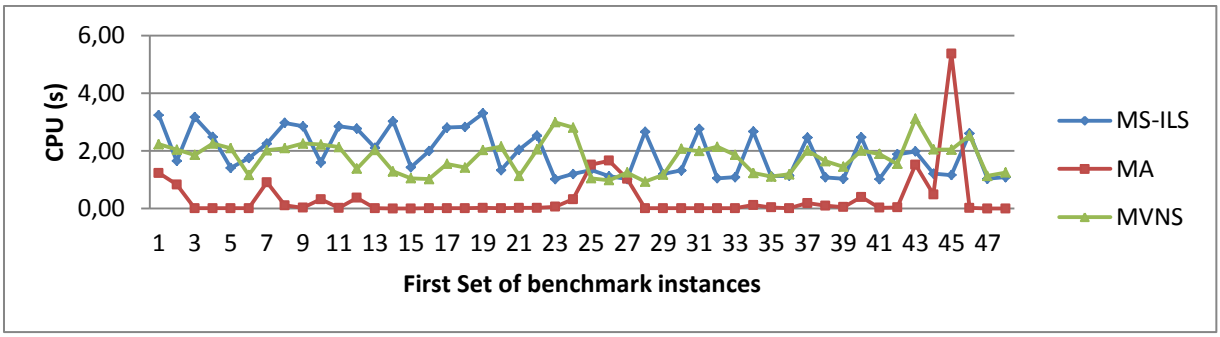

Figure IV.2. CPU time of MVNS vs MA and MS-ILS on first set of instances.

Figure IV.2 shows that both approaches solve all the instances from the first set in a very reasonable computational time. More precisely, the MVNS outperforms MS-ILS on CPU average: 1.77s with MVNS vs 1.92s with MS-ILS. Furthermore, MA outperforms MS-ILS and MVNS with a CPU average of 0.44s.

Table IV.2 shows that when dealing with the instances from the second set, using CPLEX for LGQMKP terminates with an out of memory or exceeds the time limit of 1 hour for all instances. The results show that our MVNS outperforms MS-ILS. In fact, MVNS finds the best solutions for all instances (48 instances) while MS-ILS finds the best solutions for 29 instances and MA finds the best solutions for 8 instances. More precisely, MVNS, MS-ILS and MA produce solutions with average dev of 0.001%, 0.1% and 0.51% respectively. In addition, for the instances where the average and the best results are not the same, the gaps between the best and the average results are 0.03% for MVNS, 1.36% for MS-ILS and 0.32% for MA, which proves the robustness of the MVNS.

Table IV.3. Number of the Best Results of Test Instances with n=300 for Different Parameter Levels

Parameter	Levels	MA	MS-ILS	LGQMKP	MVNS
k	10	3	15	0	**24**
	30	5	14	0	**24**
r	30	4	18	0	**24**
	150	4	11	0	**24**
d	0.25	3	13	0	**24**
	1	5	16	0	**24**

Table IV.3 provides an analysis of the number of obtained best solutions by applying CPLEX for LGQMKP, MA, MS-ILS and MVNS for each class of instances regarding the levels of parameters k, r and d. Using CPLEX for LGQMKP cannot solve large instances. In fact, no best solution nor optimal is obtained with CPLEX. The MS-ILS procedure obtains 15 best solutions for instances with 10 knapsacks and obtains 14 best solutions for instances with 30 knapsacks. Similarly, it reaches 18 best solutions for instances with 30 classes and 11 best solutions for instances with 150 classes. The MS-ILS is more successful when dealing with small instances with a low number of knapsacks, a low number of classes and a low density. The MA obtains between 3 and 5 best solutions for each parameters class, with no statistically significant difference between the parameters classes. The proposed matheuristic MVNS outperforms the MA and MS-ILS and provides the best solutions for all instances (48 small and 48 large instances).

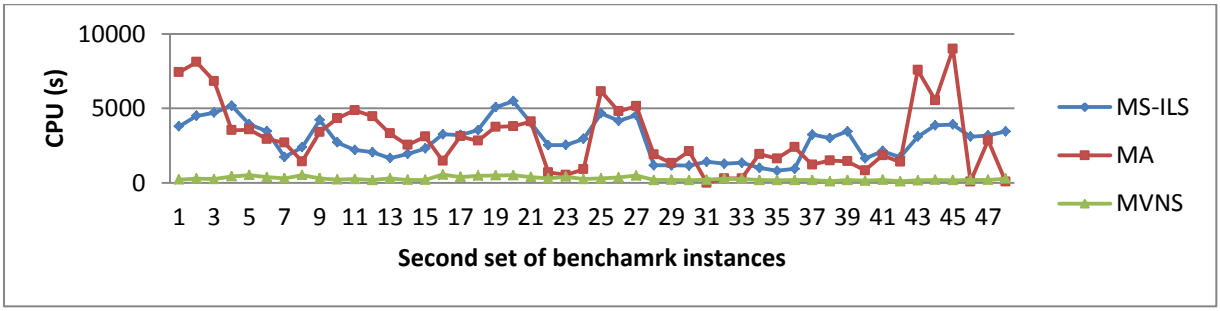

Figure IV.3. CPU time of MVNS vs MA and MS-ILS on second set of instance

Figure IV.3 shows the performance of our approach on large instances (second set) in terms of computation time. We notice that our approach is considerably faster on average than MS-ILS: 287.25 seconds and MA: 3025.75 with MVNS vs 2896.87 seconds with MS-ILS.

The proposed MVNS is more effective for large instances. The key of performance of MVNS is the iteratively decomposition of the LGQMKP into a series of less complex sub problems that may be solved in a reasonable time. This shows that combining mathematical models with metaheuristics is definitely a good option.

To better analyze the performance of the MVNS in comparison to MA and MS-ILS, we conduct an additional experiment and present the results of the paired-*t* test for the first and second set instances. Table IV.4 shows that there is no statistically significant difference between MVNS and MA and MS-ILS on quality solution for the first set of instances.

However, it has been observed that MVNS is statistically significantly different from MA and MS-ILS with mean difference equal to 175.766 and 7.123 and p-values equal to 0.032 and 0.001 respectively. This result also confirms that MVNS outperforms MA and MS-ILS for large instance. Table IV.5 of Appendix A shows that average CPU of MVNS is significantly lower than average CPU of MA and average CPU of Multi-start ILS.

Table IV.4. Results of **paired-t test** for first and second set instances on solution quality

Pairs (MVNS vs algorithm)	First set of instances		Second Set of instances	
	Mean difference	p-value	Mean difference	P-value
MVNS vs MA	8.853	0.128-	175.766	0.032+
MVNS vs MS-ILS	0	*	7.123	0.001+

*: standard error difference equal to zero; +: statistically significantly different at $\alpha = 0.05$; - : no statistically significant

Table IV.5. Results of **Tukey-test** for second set of instances on computation time CPU

Comparison set	Obj. value	P-value	Significance	Tukey result	Tukey interpretation
MA / MS-ILS / MVNS	CPU time	< 0,00	Significant difference	Factor / N / Mean / Grouping CPU (MA) / 48 / 3026 / B CPU (Multi-start ILS) / 48 / 2897 / B CPU (MVNS) / 48 / 287.2 / A	MVNS is better than MS- ILS and MA

Alpha = 0.01 Tukey Confidence = 0.99

In results of Tukey test with Alpha = 0.01 show that group B contains CPU (MA) and CPU (Multi-start ILS) and group A contains CPU (MVNS). Differences between means that share a letter are not statistically significant. CPU (MA) and CPU (Multi-start ILS) do not share a letter with CPU (MVNS), which indicates that CPU (MVNS) has a significantly lower mean than CPU (MA) and CPU (Multi-start ILS).

IV.5 Conclusion

In this chapter, we considered the Generalized Quadratic Multiple Knapsack problem with setup (GQMKP). This problem can be used to model a wide range of concrete industrial problems, including order acceptance and production scheduling. We proposed a linear formulation of the GQMKP denoted LGQMKP and a new matheuristic approach that combines VNS with IP denoted MVNS. We considered a wide set of benchmark instances to test our model LGQMKP and solving technique MVNS. The results show that only 48.9% of the instances are solved using CPLEX while considering the new model LGQMKP. The matheuristic MVNS outperforms the best algorithm in literature (MS-ILS) and provides the best solutions for all instances: the same result for 77 instances and better results for 19 instances, in a shorter computation time.

Considering the promising performance of the MVNS, an extension is expected to deal with other variants of KP such as generalized knapsack sharing problem (GKSP) and other combinatorial optimization problems involving two sets of variables.

Conclusions

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

At first, in order to draw some conclusions from the work presented in this thesis, it is necessary to draw attention to the primary goal that was considered when this research started. The primary goal was to develop cooperative approaches based upon the cooperation between neighborhood search techniques and integer programming tailored for optimizing large size instances of hard optimization problems belonging to knapsack family: linear generalized multiple knapsack problem with setup (GMKPS) and its variants such as linear MKPS, linear MCKS and quadratic variant GQMKP. In order to solve such a problem, we found two main categories:

(1) Exact methods, which try to find the best solution and prove its optimality. Indeed, due to the complexity of the considered problem, proving optimality requires a huge computational resource.

(2) (Meta-)heuristic approaches, which generate high quality solutions in a reasonable time but there is no guarantee of finding an optimal solution.

Cooperative framework, combination of exact and or (meta)heuristic methods, have emerged to solve hard optimization problems. These hybrid approaches generally provide good results since they are able to exploit simultaneously the advantages and alleviating the weaknesses of both types of methods. Thus, cooperation lead to even more powerful search models for difficult combinatorial optimization problems. In this thesis, we focused on

cooperation between variable neighborhood techniques with integer programming for solving GMKPS and its variants. Within the cooperative approaches, different mathematical programming formulations of sub problems are proposed and solved with exact solver. We have proposed three cooperative approaches can be seen as a *matheuristics* to tackle (G)MKPS, MCKS and GQMKP. The keys of better performance for our cooperative approaches instead aims to exploit the structure of the GMKPS and its variants, where the set of variables is partitioned into two levels, variables y_{rk} (classes) and variables x_{jk} (items). Thus, we decided to invest in the development of a matheuristic combining variable neighborhood techniques (VND or VNS or matheuristic VNS) and IP. The practical hardness of the problem comes from these two sets of variables that must be properly combined to reach an optimal solution. The matheuristic considers a local search technique with an adaptive perturbation mechanism to assign the classes to different knapsacks (At the same time, once the classes are chosen, the hard original problem boils down to a several classical KP) and then once the assignment is identified, applies the IP to select the items to allocate to each knapsack. Experimental results obtained on a wide set of benchmark instances clearly show the competitiveness of the proposed approach compared to CPLEX solvers and the best state-of-the-art solving techniques.

This thesis opens up several avenues for future research. They can be summarized as follows. First, it would be interesting to test the other variants of knapsack family, such as Generalized Knapsack Sharing Problem (GKSP), and also to adapt the other solution-based cooperative approaches such as cooperation between genetic algorithm (or tabu serach) and integer programming. A second perspective is to test the proposed algorithms using the different encoding schema of a program.

Further, the generalized multiple knapsack problems with setup and its variants considered in this thesis might be too simplistic compared to the real world problems that have supplementary complicated constraints or objective functions such as multi-objective scheduling problems. It is extremely expected to adapt these cooperative approaches to tackle these kinds of problems. We think that the ideas illustrated in this thesis, at least a few of them, will be useful for later research.

Conclusions

Through this thesis, we attempted to answer the primary research question: " matheuristic: exact or approximate method?". Evidently, this thesis represents a step in this research avenue and works on the subject can be pursued by considering more KP variants. In addition, I started this trip with the aim of providing additional guidelines for cooperative solution approaches for KPs. Combining matheuristics, which in some way exploit the mathematical model of a problem, is very promising and may produce effective solution approaches. I look forward to discover these future researches development, which I hope not only to observe but in some way to participate in, too.

BIBLIOGRAPHY

[1] Absi, N., and Sidhoum, S. (2008) 'The multi-item capacitated lot-sizing problem with setup times and shortage costs', *European Journal of Operational Research*, Vol. 3, pp.1351-1374.

[2] Adouani, Y., Jarboui, B., and Masmoudi M. (2018) 'A Variable Neighborhood Search with Integer Programming for the Zero-One Multiple-Choice Knapsack Problem with Setup'. In: Sifaleras A., Salhi S., Brimberg J. (eds), Variable Neighborhood Search. ICVNS 2018, *Lecture Notes in Computer Science*, Vol. 11328, pp.152-166.

[3] Adouani, Y., Jarboui, B., and Masmoudi M. (2019) 'A matheuristic for the 0-1 Generalized Quadratic Multiple Knapsack Problem'. *Optim Lett*, pp.1-22. (in press)

[4] Adouani, Y., Jarboui, B., and Masmoudi M. (2020) 'A efficient matheuristic for the Generalized Multiple Knapsack Problem with setup'. *European J. Industrial Engineering*. (in press).

[5] Akcay, Y., Li, H. and Xu, S.H. (2007) 'Greedy algorithm for the general multidimensional knapsack problem', *Annals of Operations Research*, Vol. 150, pp.17-29.

[6] Akinc, U. (2006) 'Approximate and exact algorithms for the fixed-charge knapsack problem', *European Journal of Operational Research*, Vol. 170, pp.363-375.

[7] AlMaliky, F., Hifi, M. and Mhalla, H. (2018) 'Sensitivity analysis of the setup knapsack problem to perturbation of arbitrary profits or weights', *International Transactions in Operational Research*, Vol. 25, pp.637-666.

[8] Augerat, P., Belenguer, J. M., Benavent, E., Corbern, A. and Naddef D. (1998) 'Separating capacity constraints in the CVRP using tabu search', *European Journal of Operational Research*, Vol. 106, pp.546-557.

[9] Avci, M. and Topaloglu, S. (2017) 'A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem', *Computers & Operations Research*,Vol. 83, pp.54-65.

[10] Balas, E. and Xue, J. (1996) 'Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring', *Algorithmica*, Vol. 15, pp.397-412.

[11] Bansal, J.C. and Deep, K. (2012) 'A modified binary particle swarm optimization for knapsack problems', *Applied Mathematics and Computation*, Vol. 218, pp.11042-11061.

[12] Bellman, R. (1957) 'Dynamic Programming', Princeton University Press, Princeton, New Jersey, USA.

[13] Bertsimas, D. and Tsitsiklis, J. N. (1997) 'Introduction to Linear Optimization', Athena Scientific.

[14] Bertsimas, D. and Weismantel, R. (2005) 'Optimization Over Integers', Dynamic Ideas.

[15] Billionnet, A., and Soutif, E. (2004) 'An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem', *European Journal of operational research*, Vol. 157, pp.565-575.

[16] Billionnet, A. and Soutif, E. (2004) 'Using a mixed integer programming tool for solving the 01 quadratic knapsack problem', *INFORMS Journal on Computing*, Vol. 16, pp.188-197.

[17] Burke, E.K., Li, J. and Qu, R. (2010) 'A hybrid model of integer programming and variable neighborhood search for highly-constrained nurse rostering problems', *European Journal of Operational Research*, Vol. 2003, pp.484-493.

[18] Chabrier, A., Danna, E. and Le Pape, C. (2002) 'Coopération entre génération de colonnes sans cycle et recherche locale appliquée au routage de véhicules', in: *JNPC*.

[19] Chaillou, P., Hansen, P. and Mahieu, Y. (1989) 'Best network ow bounds for the quadratic knapsack problem', *Combinatorial Optimization*, Vol. 1403, pp.225-235.

[20] Chajakis, E.D. and Guignard, M. (1994) 'Exact algorithms for the setup knapsack problem', *INFOR*, Vol. 32, pp.124-142.

[21] Chebil, K. and Khemakhem, M. (2015) 'A dynamic programming algorithm for the knapsack problem with setup', *Computers & Operations Research*, Vol. 64, pp.40-50.

[22] Chen, P., Huang, H. and Dong, X.Y. (2010) 'Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem', *Expert Systems with Applications*, Vol. 37, pp.1620-1627.

[23] Chen, Y. and Hao, J. K. (2016) 'Memetic search for the generalized quadratic multiple knapsack problem', *IEEE Transactions on Evolutionary Computation*, Vol. 20, pp.908-923.

[24] Cotta, C., Aldana, J. F., Nebro, A.J. and Troya, J. M. (1995) 'Hybridizing genetic algorithms with branch and bound techniques for the resolution of the tsp', in: D.W. Pearson, N.C. Steele, R.F. Albrecht (Eds.), *Artificial Neural Nets and Genetic Algorithms* 2, Springer-Verlag, Vol. 1995, pp.277-280.

[25] Crowder, H., Johnson, E.L. and Padberg, M. (1983) 'Solving large-scale zero-one linear programming problems', *Operations Research*, Vol. 31, pp. 803-834.

[26] Chu, P. and Beasley, J. (1998) 'A Genetic Algorithm for the Multidimensional Knapsack Problem', *Journal of Heuristics*, Vol. 4, pp.63-86.

[27] Dantzig, G. B. and Thapa, M. N. (1997) 'Linear Programming 1: Introduction', *Springer-Verlag, New York*.

[28] Dantzig, G. B. and Thapa, M. N. (2003) 'Linear Programming 2: Theory and Extensions', *Springer-Verlag, New York*.

[29] Dantzig, G. B. (2002) 'Linear programming', *Operations Research*, Vol. 50, pp.42-47.

[30] Dasgupta, S., Papadimitriou, C.H. and Vazirani, U (2006) 'Algorithms', McGraw-Hill.

[31] Dantzig, G. B. (1957) 'Discrete variable extremum problems', *Operation Research*, Vol. 5, pp.266-277.

[32] Della, C.F., Salassa, F. and Scatamacchia, R. (2017) 'An exact approach for the 0-1 knapsack problem with setups', *Computers & Operations Research*, Vol. 80, pp.61-67.

[33] Dogan, N., Bilgiçer, K and Saraç, T. (2012) 'Quadratic multiple knapsack problem with setups and a solution approach', In Proceedings of the International Conference on *Industrial Engineering and Operations Management*, Turkey, pp.3-6.

[34] Duarte, A., Pantrigo, J. J., Pardo, E.G. and Mladenovic, N. (2015) 'Multi-objective variable neighborhoodsearch: an application to combinatorial optimization problems', *J. Glob. Optim*, Vol. 63, pp.515-536.

[35] Dudziński, K. and S. Walukiewicz, S. (1987) 'Exact methods for the knapsack problem and its generalizations', *European Journal of Operational Research*, Vol. 28, pp.3-21.

[36] Dumitrescu, I. and Stützel, T. (2003) 'Combinations of local search and exact algorithms', *Applications of evolutionary computation*, Vol. 2611, pp.211-223.

[37] Dumitrescu, I. and Stützle, T. (2009) 'Usage of Exact Algorithms to Enhance Stochastic Local Search Algorithms', In Maniezzo, V., Stützle, T., and Voß, S. editors,

Matheuristics Hybridizing Metaheuristics and Mathematical Programming, of Annals of Information Systems, Vol. 10, pp.103-134. *Springer, New York, NY*.

[38] Erromdhani, R., Jarboui, B., Eddaly, M., Rebai, A. and Mladenovic, N. (2017) 'variable neighborhood formulation search approach for the multi-item capacitated lot-sizing problem with time windows and setup times', *Yugoslav Journal of Operations Research*, Vol. 27, pp. 301–322.

[39] Fernandes, S. and Lourenco, H. (2007) 'Hybrid combining local search heuristics with exact algorithms', in: Almeida, F. et al. (Eds.), V Congreso Espanolsobre Metaheursticas. *Algoritmos Evolutivos y Bioinspirados*, Spain, pp.269-274.

[40] Fleszar, K., Charalambous, C. and Hindi, K.S. (2012) 'A variable neighborhood descent heuristic for the problem of make span minimization on unrelated parallel machines with setup times', *Journal of Intelligent*, Vol. 23, pp.1949-1958.

[41] Fischetti, M. and Lodi, A. (2003) 'Local Branching. *Mathematical Programming Series B*', Vol 98, pp.23-47.

[42] Freville, A. and Plateau, G. (1986) 'Heuristics and reduction methods for multiple constraints 0-l linear programming problems', *European Journal of Operational Research*, Vol. 24, pp.206-215.

[43] Garcia-Martinez, C., Rodriguez, F. J. and Lozano, M. (2014) 'Tabu-enhanced iterated greedy algorithm: A case study in the quadratic multiple knapsack problem', *European Journal of Operational Research*, Vol. 232, pp.454-463.

[44] Ghasemi, T. and Razzazi, M. (2011) 'Development of core to solve the multidimensional multiple-choice knapsack problem', *Computers and Industrial Engineering*, Vol. 60, pp.349-360.

[45] Gomory, R. (1958) 'Outline of an algorithm for integer solutions to linear programs', Bulletin of the American Mathematical Society, Vol. 64, pp.275-278.

[46] Glover, F. (1989) 'Tabu search-part I', *ORSA Journal on computing*, Vol. 1, pp.190-206.

[47] Goldberg, D.E. (1989) 'Genetic algorithms in search, optimization, and machine learning', Addison-WesleyLongman Publishing Co.

[48] Haddar, B., Khemakhem, M., Rhimi, H. and Chabchoub, H. (2016) 'A quantum particle swarm optimization for the 0-1 generalized knapsack sharing problem', *Natural Computing*, Vol 15, pp.153-164.

[49] Hanafi, S. and Fréville, A. (1998) 'An efficient tabu search approach for the 0-1 multidimensional knapsack problem, *European Journal of Operational Research*, Vol. 106, pp. 659-675.

[50] Hanafi, S., Lazić, J., Mladenović, N. and Wilbaut, C. (2010) 'New hybrid matheuristics for solving the multidimensional knapsack problem', *International Workshop on Hybrid Metaheuristics, Springer*, Vol. 6373, pp.118-132.

[51] Hansen, P., Mladenovic, N. and Moreno Perez, J. A. (2010) 'Variable neighborhood search: methods and applications', *Annals of Operations Research*, Vol. 175, pp.367-407.

[52] Hansen, P., Mladenovic, N., Todosijevic, R. and Hana, S. (2017) 'Variable neighborhood search: basics and variants', *EURO Journal on Computational Optimization*, Vol. 5, pp.423-454.

[53] Hana, S., Lazic, J., Mladenovic, N. and Wilbaut, C. (2010) 'New hybrid matheuristics for solving the multidimensional knapsack problem', *International Workshop on Hybrid Metaheuristics. Springer*. Vol. 6373, pp.118-132.

[54] Hifi, M., Michrafy, M. and Sbihi, A. (2006) 'A reactive local search-based algorithm for the multiple-choice multidimensional knapsack problem', *Computational Optimization and Applications*, Vol 33, pp.271-285.

[55] Hiley, A. and Julstrom, B. A. (2006) 'The quadratic multiple knapsack problem and three heuristic approaches to it', In Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp.547-552.

[56] Horowitz, E. and Sahni, S. (1974) 'Computing partitions with applications to the knapsack problem', *Journal of the ACM* (JACM), Vol. 21, pp.277-292.

[57] Jarboui, B., Derbel, H., Hanafi, S. and Maldenovic, N. (2013) 'Variable neighborhood search for location routing', *Computers & Operations Research*, Vol. 40, pp.47-57.

[58] Johnson, E., Mehrotra, A. and Nemhauser, G. (1993) 'Min-cut clustering', *Math. Programming*. Vol. 62, pp.133-152.

[59] Jourdan, L., Basseur, M. and Talbi, E.G. (2009) 'Hybridizing exact methods and metaheuristics', *European Journal of Operational Research*, Vol. 199, pp.620-629.

[60] Jünger, M., Liebling, T., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G. and Wolsey, L. A., editors (2009) '50 Years of Integer Programming 1958–2008', Springer.

[61] Kellerer, H., Pferschy, U., Pisinger, D. (2004) 'Knapsack Problems', Springer.

[62] Kennedy, J. and Eberhart, R. (1995) 'Particle Swarm Optimization', *IEEE International Conference on Neural Net- work*, Vol. 4, pp.1942-1948.

[63] Khemakhem, M. and Chebil, K. (2016) 'A tree search based combination heuristic for the knapsack problem with setup', *Computers and Industrial Engineering*, Vol. 99, pp.280-286.

[64] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983). 'Optimization by Simulated Annealing', *Science*.

[65] Klee, V. and Minty, G. J, (1972) 'How good is the simplex algorithm', In O. Shisha, editor, Inequalities, Academic Press, New York, Vol. 3, pp.159-175.

[66] Klepeis, J. L., Pieja, M. J. and Floudas, C.A. (2003) 'Hybrid global optimization algorithms for protein structure prediction: Alternating hybrids', *Biophysical Journal*, Vol. 4, pp.869-882.

[67] Kolesar, P. J. (1967) 'A branch and bound algorithm for the knapsack problem', *Management Science*, Vol. 13, pp.723-735.

[68] Kong, X., Gao, L., Ouyang, H. and Li, S. (2015) 'A simplifed binary harmony search algorithm for large scale 0-1 knapsack problems', *Expert Systems with Applications*, Vol. 42, pp.5337-5355.

[69] Kostikas, K. and Fragakis, C. (2004) 'Genetic programming applied to mixed integer Programming', in: EuroGP 2004, *Lecture Notes in Computer Science*, Vol. 3003, pp. 113-124.

[70] Lahyani R., Coelho, L.C., Khemakhem, M., Laporte, G., Semet, F. (2015) 'A multi-compartment vehicle routing problem arising in the collection of olive oil in Tunisia', In: *Omega*, Vol. 51, pp.1-10.

[71] Lamghari, A., Dimitrakopoulos, R. and Ferland, J.A. (2015) 'A hybrid method based on linear programming and variable neighborhood descent for scheduling production in open-pit mines', *Journal of Global Optimization*, Vol. 63, pp.555-582.

[72] Land, A. H. and Doig, A. G. (1960) 'An automatic method of solving discrete programming problems', *Econometrica*: *Journal of the Econometric Society*, Vol. 28, pp.497-520.

[73] Lourenço, H. R., Martin, O. C. and Stützle, T. (2010) 'Iterated Local Search: Framework and Applications', Springer US, Vol. 146, pp.363-397.

[74] Maniezzo, V., Stutzle, T. and Voss, S. (2009) 'Matheuristics: Hybridizing Metaheuristics and Mathematical Programming, *Annals of Information Systems*, Vol. 10, Springer, Heidelberg.

[75] Martello, S. and Toth, P. (1977) 'An upper bound for the zero-one knapsack problem and a branch and bound algorithm', *European Journal of Operational Research*, Vol. 1, pp.169-175.

[76] Martello, S. and Toth, P. (1980) 'Solution of the zero-one multiple knapsack problem', *European Journal of Operational Research*, Vol. 4, pp.276-283.

[77] Martello, S. and Toth, P. (1990) 'Knapsack problems: Algorithms and computer implementations', 605 Third Avenue, New York, NY 10158-0012, USA: John Wiley & Sons.

[78] Martello, S., Pisinger, D. and Toth, P. (2000) 'New trends in exact algorithms for the 0-1 knapsack problems', *European Journal of Operational Research*, Vol. 123, pp.325-332.

[79] McLay, L. (2006) 'Designing aviation security systems: Theory and practice', Dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

[80] Michel, S., Perrot, N. and Vanderbeck, F. (2009) 'Knapsack problems with setups', *European Journal of Operational*, Vol. 196, pp.909-918.

[81] Mladenović, N. and Hansen, P. (1997) 'Variable neighborhood search'. *Computers & Operation Research*, Vol 24, pp.1097-1100.

[82] Nemhauser, G. L. and Wolsey, L. A. (1988) 'Integer and Combinatorial Optimization', John Wiley & Sons, New York, NY, USA.

[83] Osman, I. H. and Laporte, G. (1996) 'Metaheuristics: A bibliography', *Annals of Operations research*, Vol. 63, pp.511-623.

[84] Peng, B., Liu, M., L, Z., Kochengber, G. and Wang, H. (2016) 'An Ejection Chain Approach for the Quadratic Multiple Knapsack Problem', *European Journal of Operational Research.*

[85] Penna, P. H., Subramanian, A. and Ochi, L.S . (2013) 'An Iterated Local Search heuristic for the Heterogeneous Fleet Vehicle Routing Problem', *Journal of heuristics*, Vol. 19, pp.201-232.

[86] Peter, J. K. (1967) 'A branch and bound algorithm for the knapsack problem', *Management Science*, Vol. 13, pp.723-735.

[87] Pferschy, U. and Rosario, S. (2018) 'Improved dynamic programming and approximation results for the knapsack problem with setups', *International Transactions in Operational Research*, Vol 25, pp.667-682.

[88] Pisinger, D. (1999) 'An exact algorithm for large multiple knapsack problems', *European Journal of Operational Research*, Vol. 114, pp.528-541.

[89] Pisinger, D. (2007) 'The quadratic knapsack problem-a survey', *Discrete Applied Mathematics*. Vol. 155, pp.623-48.

[90] Prandtstetter, M. and Raidl, G.R. (2008) 'An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem', *European Journal of Operational Research*, Vol. 191, pp.1004-1022.

[91] Puchinger, J. and Raidl, G.R. (2005) 'Combining meta-heuristics and exact algorithms in combinatorial optimization', in: Mira, J. and Alvarez, J.R. (Eds.), Artificial Intelligence and Knowledge Engineering Applications. Berlin Heidelberg, Vol. 3562, pp.41-53.

[92] Qin, J., Xu, X., Wu, Q. and Cheng, T. C. E. (2016) 'Hybridization of tabu search with feasible and infeasible local searches for the quadratic multiple knapsack problem', *Computers & Operations Research*, Vol. 66, pp.199-214.

[93] Richard, L. and Eleftherios, M. (1979) 'New Greedy-like Heuristics for the Multidimensional 0-1 Knapsack Problem', *Operations Research*, Vol. 27, pp.1101-1114.

[94] Sara, T. and Sipahioglu, A. (2014) 'Generalized quadratic multiple knapsack problem and two solution approaches', *Computers & Operations Research*, Vol. 43, pp.78-89.

[95] Sinha, A. and Zoltners, A. A. (1979) 'The multiple-choice knapsack problem', *Operations Research*, Vol. 27, pp.503-515.

[96] Soyster, A. L., Lev, B. and Slivka, W. (1978) 'Zero-one programming with many variables and few constraints', *European Journal of Operational Research*, Vol. 2, pp.195-201.

[97] Stützle, T.G. (1998) 'Local search algorithms for combinatorial problems: analysis, improvements, and new applications, Infix.

[98] Sundar, S. and Singh, A. (2010) 'A swarm intelligence approach to the quadratic multiple knapsack problem', *Neural Information Processing, Theory and Algorithms*, pp.626-633.

[99] Tlili, T., Yahyaoui, H. and Krichen, S. (2016) 'An iterated variable neighborhood descent hyperheuristic for the quadratic multiple knapsack problem', *Software Engineering, Artificial Intelligence*, Vol. 612, pp.245-251.

[100] Vasquez, M. and Hao, J. K. (2001) 'A hybrid approach for the 0-1 multidimensional knapsack problem', in: Proceedings of the International Joint Conference on Artificial Intelligence, Washington, pp.328-333.

[101] Visee, M., Teghem, J., Pirlot, M. and Ulungu, E.L. (1998) 'Two-phases Method and Branch and Bound Procedures to Solve the Biobjective Knapsack Problem', *J. Glob. Optim*, Vol. 12, pp.139-155.

[102] Wen, M., Iwamura, K. (2008) 'Facility location–allocation problem in random fuzzy environment: Using (α, β)-cost minimization model under the Hurewicz criterion', *Computers & Mathematics with Applications*, Vol. 55, pp. 704-713

[103] Yang, Y. and Bulfin, R.L. (2009) 'An exact algorithm for the knapsack problem with setup', *Int. J. Operational Research*, Vol. 5, pp.280-291.

[104] Yang, Y. (2006) 'Knapsack problems with setup', Dissertation, Auburn university.

[105] Zhang, C.W. and Ong, H.L. (2004) 'Solving the biobjective zero-one knapsack problem by an efficient LP-based heuristic', *European Journal of Operational Research*, Vol. 159,pp.545-557.

Appendix A

In this appendix. We provide the detailed results of our computational experiments for GMKPS, MKPS and MCKS. The following notations are considered in the all tables:

- $Instance$: Number of the instance
- IP_{stat}:
  - 101: Optimal solution found.
  - 107: Time limit exceeded.
  - 109: Out of memory.

Detailed computational results for GMKPS

Table A.1: Detailed computational results for **GMKPS** with $n_i \in [40.60]$

T	N	$instance$	IP_{OBJ}	IP_{CPU}	IP_{stat}	$VND\&IP_{OBJ}$	$VND\&IP_{CPU}$	Gap (%)
5	10	1	812913	3600.086	107	812913	3.178	0.0000
		2	958546	3600.055	107	958547	10.306	-0.0001
		3	960121	3600.263	107	960121	3.27	0.0000
		4	676465	3600.147	107	676466	2.875	-0.0001
		5	955234	3600.103	107	955234	2.919	0.0000
		6	669278	3600.013	107	669285	3.341	-0.0010
		7	668532	3600.039	107	668538	13.622	-0.0009
		8	612232	3600.032	107	612235	1.529	-0.0005
		9	685762	3600.077	107	685764	4.756	-0.0003
		10	600841	1015.669	101	600841	3.054	0.0000
	20	1	1001475	3600.186	107	1001475	1.446	0.0000
		2	628864	3600.123	107	628864	1.655	0.0000
		3	674355	3600.199	107	674343	12.27	0.0018
		4	918838	3600.052	107	918840	5.614	-0.0002
		5	936002	3600.063	107	936002	4.405	0.0000
		6	779864	3600.116	107	779867	6.604	-0.0004

		7	920233	3600.089	107	920233	3.247	0.0000
		8	682760	3600.106	107	682760	1.092	0.0000
		9	591179	3600.109	107	591179	2.098	0.0000
		10	776047	3600.158	107	776050	5.933	-0.0004
	30	1	816917	3600.129	107	816917	1.842	0.0000
		2	1009108	3600.143	107	1009108	2.027	0.0000
		3	1028611	3600.339	107	1028614	6.09	-0.0003
		4	1098279	3600.387	107	1098280	1.866	-0.0001
		5	829696	3600.166	107	829685	2.996	0.0013
		6	1013232	3600.194	107	1013233	2.322	-0.0001
		7	760349	38.049	101	760349	2.438	0.0000
		8	905098	3600.321	107	905104	1.938	-0.0007
		9	753433	3600.231	107	753433	1.703	0.0000
		10	924785	3600.122	107	924787	2.021	-0.0002
10	10	1	1603852	3600.223	107	1603954	12.533	-0.0064
		2	1751367	3600.03	107	1755536	3.815	-0.2380
		3	1630714	3600.13	107	1636663	17.728	-0.3648
		4	1399234	3600.065	107	1401813	30.325	-0.1843
		5	1892218	3600.043	107	1892365	5.82	-0.0078
		6	1371290	3600.061	107	1373297	15.32	-0.1464
		7	1211435	3600.281	107	1211432	40.47	0.0002
		8	1216160	3600.088	107	1216286	30.495	-0.0104
		9	1244960	3600.025	107	1247261	31.87	-0.1848
		10	1132371	3600.053	107	1132349	40.299	0.0019
	20	1	1871189	3600.223	107	1871292	1.606	-0.0055
		2	1292390	3600.27	107	1293785	16.059	-0.1079
		3	1477595	3600.062	107	1479116	6.489	-0.1029
		4	1682582	3601.194	107	1682672	6.168	-0.0053
		5	1974522	3600.063	107	1974552	4.173	-0.0015
		6	1694731	3600.043	107	1694830	5.568	-0.0058
		7	1891201	3600.264	107	1891514	7.913	-0.0166
		8	1402244	3600.188	107	1402275	3.853	-0.0022
		9	1175772	3600.036	107	1175840	30.568	-0.0058
		10	1449121	3600.229	107	1449615	18.501	-0.0341
	30	1	1612291	3600.297	107	1613746	5.056	-0.0902
		2	2031808	3600.243	107	2031831	6.206	-0.0011
		3	1958030	3600.327	107	1958088	7.968	-0.0030
		4	2165244	3600.298	107	2165333	3.074	-0.0041
		5	1669904	3600.23	107	1669857	5.269	0.0028

		6	2125487	3600.193	107	2125488	4.746	0.0000
		7	1681264	3600.235	107	1681310	10.97	-0.0027
		8	1960791	3600.208	107	1962504	8.096	-0.0874
		9	1750652	3600.28	107	1750834	7.827	-0.0104
		10	1822033	3600.314	107	1822047	5.646	-0.0008
15	10	1	2288605	3601.193	107	2319142	14.203	-1.3343
		2	2265070	3600.068	107	2290209	11.028	-1.1099
		3	2258467	3600.123	107	2280258	70.563	-0.9649
		4	2032309	3600.054	107	2049983	66.671	-0.8697
		5	2312884	3600.955	107	2312884	3.875	0.0000
		6	2036890	3600.06	107	2046672	60.437	-0.4802
		7	1803636	3600.068	107	1807162	65.092	-0.1955
		8	1797939	3600.776	107	1817259	67.769	-1.0746
		9	1747982	3600.026	107	1774638	40.749	-1.5250
		10	1678324	3600.09	107	1683655	43.905	-0.3176
	20	1	2706056	3600.325	107	2706826	6.681	-0.0285
		2	1794053	3600.238	107	1798826	51.203	-0.2660
		3	2187871	3600.2	107	2195401	19.477	-0.3442
		4	2469025	3600.078	107	2476018	18.337	-0.2832
		5	2908199	3600.218	107	2910999	7.59	-0.0963
		6	2372053	3600.065	107	2380982	8.745	-0.3764
		7	2698645	3600.155	107	2701151	9.817	-0.0929
		8	1901903	3600.242	107	1916093	31.402	-0.7461
		9	1653612	3600.047	107	1653975	69.014	-0.0220
		10	2122781	3600.35	107	2129807	34.641	-0.3310
	30	1	2684661	3600.404	107	2689634	12.826	-0.1852
		2	2858448	3600.307	107	2845417	53.431	0.4559
		3	2725434	3600.103	107	2725778	6.382	-0.0126
		4	3003857	3600.409	107	3006413	7.491	-0.0851
		5	2473755	3602.791	107	2475063	98.043	-0.0529
		6	3142366	3601.627	107	3149552	11.535	-0.2287
		7	2636996	3600.52	107	2637392	10.57	-0.0150
		8	3035371	3600.638	107	3035613	4.362	-0.0080
		9	2718106	3600.536	107	2719783	6.126	-0.0617
		10	2719897	3600.638	107	2727728	7.769	-0.2879
20	10	1	2324170	3607.381	107	2337831	8.174	-0.5878
		2	2309245	3600.027	107	2331711	13.281	-0.9729
		3	2351799	3600.03	107	2381083	9.554	-1.2452
		4	2261308	3600.348	107	2297629	7.533	-1.6062

		5	2283262	3600.059	107	2297309	4.16	-0.6152
		6	2252182	3600.025	107	2267316	9.184	-0.6720
		7	2150880	3600.361	107	2184523	15.435	-1.5642
		8	2046575	3602.922	107	2062592	32.017	-0.7826
		9	2261341	3600.095	107	2296965	40.602	-1.5753
		10	2097028	3601.874	107	2119150	45.593	-1.0549
	20	1	3781192	3600.119	107	3795371	60.352	-0.3750
		2	2288307	3600.1	107	2314756	60.982	-1.1558
		3	2825909	3600.092	107	2850146	60.916	-0.8577
		4	3265962	3600.102	107	3302637	50.649	-1.1229
		5	3714754	3600.068	107	3730447	17.406	-0.4225
		6	3238651	3600.162	107	3264119	30.465	-0.7864
		7	3467812	3600.041	107	3525277	35.235	-1.6571
		8	2384775	3600.416	107	2405080	50.82	-0.8514
		9	2126565	3600.165	107	2151009	44.046	-1.1495
		10	2681315	3600.567	107	2699665	55.606	-0.6844
	30	1	3549246	3600.166	107	3562054	16.199	-0.3609
		2	3657501	3600.123	107	3659987	34.552	-0.0680
		3	3622280	3600.36	107	3639456	18.721	-0.4742
		4	3861616	3600.147	107	3886415	9.24	-0.6422
		5	3501286	3600.453	107	3519183	16.146	-0.5112
		6	4024439	3600.07	107	4065645	8.873	-1.0239
		7	3342156	3600.121	107	3347683	23.877	-0.1654
		8	3844833	3601.082	107	3861978	12.41	-0.4459
		9	3720185	3600.165	107	3728062	12.421	-0.2117
		10	3495901	3600.798	107	3512312	12.613	-0.4694

Table A.2: Detailed computational results for **GMKPS** with $n_i \in [60.90]$

T	N	$Instance$	IP_{OBJ}	IP_{CPU}	IP_{stat}	$VND\&IP_{OBJ}$	$VND\&IP_{CPU}$	Gap (%)
5	10	1	1135990	3600.374	107	1135990	1.874	0.0000
		2	1309346	3600.031	107	1309351	2.574	-0.0004
		3	838462	3600.167	107	838484	5.747	-0.0026
		4	1338915	3600.045	107	1338922	7.22	-0.0005
		5	1047773	3600.266	107	1047782	8.833	-0.0009
		6	1089294	3600.105	107	1089295	9.293	-0.0001
		7	1353672	3600.058	107	1353659	10.97	0.0010
		8	1153287	3600.068	107	1153287	4.812	0.0000

		9	869128	3600.121	107	869128	3.736	0.0000
		10	1529682	3600.173	107	1529682	2.005	0.0000
	20	1	1099777	190.883	101	1099777	3.115	0.0000
		2	1323126	3600.126	107	1323130	2.233	-0.0003
		3	1329746	3600.093	107	1329746	3.902	0.0000
		4	997809	3600.073	107	997810	8.285	-0.0001
		5	1293494	3600.515	107	1293496	6.468	-0.0002
		6	1422106	3600.028	107	1422107	3.602	-0.0001
		7	866454	3600.172	107	866460	6.847	-0.0007
		8	1575520	3600.124	107	1575520	1.601	0.0000
		9	1252189	3600.059	107	1252189	1.148	0.0000
		10	1119950	3600.168	107	1119956	3.701	-0.0005
	30	1	973907	3600.188	107	973907	2.542	0.0000
		2	1154917	3600.091	107	1154933	2.628	-0.0014
		3	1179293	3600.098	107	1179300	2.776	-0.0006
		4	1044345	3600.007	107	1044345	3.965	0.0000
		5	1178724	3600.147	107	1178724	2.148	0.0000
		6	1284699	332.299	101	1284699	4.272	0.0000
		7	1259316	3600.345	107	1260925	4.028	-0.1278
		8	1136234	3600.279	107	1136247	3.255	-0.0011
		9	915931	3600.038	107	915938	1.522	-0.0008
		10	1320349	287.946	101	1320349	1.097	0.0000
10	10	1	2203048	3600.007	107	2204837	8.977	-0.0812
		2	2641149	3601.009	107	2644369	6.032	-0.1219
		3	1783667	3600.145	107	1783114	40.723	0.0310
		4	2778356	3600.036	107	2778490	5.28	-0.0048
		5	2295233	3600.215	107	2295319	11.439	-0.0037
		6	1956384	3600.047	107	1969663	30.877	-0.6788
		7	2736305	3600.363	107	2740557	5.188	-0.1554
		8	2431150	3600.118	107	2441336	3.955	-0.4190
		9	1894367	3600.542	107	1897986	30.702	-0.1910
		10	2923308	3600.044	107	2923434	4.275	-0.0043
	20	1	2088653	3600.106	107	2091981	13.178	-0.1593
		2	2528469	3600.336	107	2532270	3.461	-0.1503
		3	2836970	3600.25	107	2836991	3.872	-0.0007
		4	1956109	3600.195	107	1960748	13.905	-0.2372
		5	2595511	3600.271	107	2600713	4.548	-0.2004
		6	2530249	3600.646	107	2551291	3.356	-0.8316
		7	1989904	3600.255	107	2000177	31.382	-0.5163

		8	3223699	3600.172	107	3228096	3.437	-0.1364
		9	2615473	3600.69	107	2615544	2.907	-0.0027
		10	2311150	3600.058	107	2311189	5.131	-0.0017
	30	1	2350838	3600.406	107	2352883	3.639	-0.0870
		2	2327171	3600.236	107	2327204	4.433	-0.0014
		3	2260463	3600.336	107	2262071	4.175	-0.0711
		4	2165817	3600.28	107	2172867	5.626	-0.3255
		5	2269674	3600.328	107	2275439	7.344	-0.2540
		6	2513193	3600.345	107	2513215	9.595	-0.0009
		7	2280289	3600.312	107	2280865	12.213	-0.0253
		8	2278849	3600.259	107	2278915	9.181	-0.0029
		9	1906991	3600.27	107	1907004	15.408	-0.0007
		10	2492253	3600.675	107	2492587	8.239	-0.0134
15	10	1	3255164	3600.348	107	3304096	90.63	-1.5032
		2	3473823	3600.032	107	3518744	10.681	-1.2931
		3	2654033	3600.662	107	2691244	61.415	-1.4021
		4	3487442	3603.284	107	3502706	7.297	-0.4377
		5	3244253	3601.19	107	3292369	70.576	-1.4831
		6	3064391	3600.047	107	3108121	64.767	-1.4270
		7	3360001	3600.093	107	3387869	6.991	-0.8294
		8	3128586	3600.24	107	3159139	9.691	-0.9766
		9	2757744	3600.036	107	2784983	67.011	-0.9877
		10	3558666	3600.028	107	3577577	41.379	-0.5314
	20	1	3006366	3600.956	107	3014696	43.424	-0.2771
		2	3865636	3600.309	107	3865508	12.672	0.0033
		3	4124886	3600.12	107	4137427	7.557	-0.3040
		4	2881730	3600.3	107	2903412	51.953	-0.7524
		5	3747503	3600.079	107	3768181	21.949	-0.5518
		6	3735810	3600.063	107	3740469	18.468	-0.1247
		7	3140047	3600.341	107	3141879	32.078	-0.0583
		8	4693676	3600.301	107	4721239	10.162	-0.5872
		9	3926113	3600.451	107	3948143	14.057	-0.5611
		10	3475762	3600.327	107	3494340	17.407	-0.5345
	30	1	3279295	3600.405	107	3282773	32.283	-0.1061
		2	3563092	3600.883	107	3569378	19.79	-0.1764
		3	3446242	3601.476	107	3448317	13.086	-0.0602
		4	3459757	3600.68	107	3477683	29.574	-0.5181
		5	3376620	3600.306	107	3393784	21.716	-0.5083
		6	3757424	3600.858	107	3763712	12.005	-0.1673

		7	3341049	3600.534	107	3341309	12.518	-0.0078
		8	3429455	3601.332	107	3435875	16.399	-0.1872
		9	2719748	3600.781	107	2744656	93.617	-0.9158
		10	3733701	3600.646	107	3740622	15.888	-0.1854
20	10	1	3370542	3600.156	107	3380053	26.79	-0.2822
		2	3516741	3600.055	107	3531702	18.601	-0.4254
		3	3377210	3600.095	107	3429730	112.481	-1.5551
		4	3503338	3600.062	107	3518703	10.255	-0.4386
		5	3473398	3600.081	107	3532934	25.196	-1.7141
		6	3349932	3600.053	107	3372946	26.573	-0.6870
		7	3411103	3600.076	107	3418334	14.85	-0.2120
		8	3198627	3600.022	107	3202779	7.903	-0.1298
		9	3194611	3600.077	107	3244499	24.031	-1.5616
		10	3600971	3600.037	107	3608919	10.182	-0.2207
	20	1	3840225	3600.451	107	3887847	72.028	-1.2401
		2	5119179	3600.097	107	5128914	30.924	-0.1902
		3	4983417	3600.101	107	5044642	35.952	-1.2286
		4	3917960	3600.097	107	3949932	61.478	-0.8160
		5	4972891	3600.335	107	5021380	39.482	-0.9751
		6	4997562	3600.25	107	5068629	44.855	-1.4220
		7	4120857	3600.092	107	4150699	59.168	-0.7242
		8	6059568	3600.077	107	6114417	9.797	-0.9052
		9	5012324	3600.579	107	5053860	38.894	-0.8287
		10	4498643	3600.05	107	4527271	60.816	-0.6364
	30	1	4402634	3600.194	107	4444768	44.05	-0.9570
		2	4748304	3601.264	107	4756582	31.391	-0.1743
		3	4341344	3601.488	107	4374171	31.573	-0.7561
		4	4554378	3600.128	107	4602355	41.018	-1.0534
		5	4471200	3600.694	107	4535788	51.538	-1.4445
		6	5000692	3600.051	107	5046797	23.562	-0.9220
		7	4314826	2506.813	109	4355036	44.634	-0.9319
		8	4374905	3600.052	107	4408044	54.692	-0.7575
		9	3701426	3600.242	107	3753223	65.544	-1.3994
		10	4722127	3600.228	107	4752095	25.673	-0.6346

Table A.3: Detailed computational results for **GMKPS** with $n_i \in [90.110]$

T	N	$Instance$	IP_{OBJ}	IP_{CPU}	IP_{stat}	$VND\&IP_{OBJ}$	$VND\&IP_{OBJ}$	$Gap\ (\%)$
5	10	1	1867438	3600.077	107	1867448	7.014	-0.0005

		2	1674725	3600.42	107	1674727	2.198	-0.0001
		3	1759784	3600.188	107	1759784	1.986	0.0000
		4	1515969	3600.183	107	1515973	3.986	-0.0003
		5	1730973	3600.019	107	1730990	4.488	-0.0010
		6	1408926	3600.276	107	1408929	3.95	-0.0002
		7	1692385	3600.266	107	1692386	2.366	-0.0001
		8	1528499	3600.158	107	1528500	2.058	-0.0001
		9	1530607	3600.053	107	1530612	3.535	-0.0003
		10	1540665	3600.057	107	1540670	2.427	-0.0003
	20	1	1604647	3600.468	107	1604648	5.591	-0.0001
		2	1737908	3600.102	107	1737941	6.432	-0.0019
		3	1657099	3600.297	107	1657101	6.228	-0.0001
		4	1551046	3600.132	107	1551052	3.842	-0.0004
		5	1378593	3600.111	107	1378600	3.452	-0.0005
		6	1657266	3600.301	107	1657270	3.268	-0.0002
		7	2142847	1502.733	101	2142847	5.913	0.0000
		8	1388770	3600.127	107	1388781	4.047	-0.0008
		9	1385251	3600.095	107	1385258	4.954	-0.0005
		10	1656632	3600.133	107	1656634	4.047	-0.0001
	30	1	1531023	3600.055	107	1531024	6.069	-0.0001
		2	1499973	3600.307	107	1499985	3.591	-0.0008
		3	1526817	3600.413	107	1527417	1.394	-0.0393
		4	1712118	3600.176	107	1712120	5.536	-0.0001
		5	1703228	3600.107	107	1703232	4.054	-0.0002
		6	1621348	21.249	101	1621348	5.554	0.0000
		7	1950132	3600.153	107	1950132	2.54	0.0000
		8	1956198	3600.285	107	1956204	1.284	-0.0003
		9	1846935	3600.055	107	1846940	3.681	-0.0003
		10	1689929	3600.62	107	1689932	6.068	-0.0002
10	10	1	3224526	3600.063	107	3230187	13.885	-0.1756
		2	3094088	3600.02	107	3106717	16.861	-0.4082
		3	3540100	3600.049	107	3545610	9.479	-0.1556
		4	2994702	3600.061	107	2994730	30.617	-0.0009
		5	3347214	3600.029	107	3347290	9.281	-0.0023
		6	2917362	3600.052	107	2928397	7.34	-0.3783
		7	2848914	3600.03	107	2850447	25.964	-0.0538
		8	3138287	3596.102	107	3138789	21.639	-0.0160
		9	2898892	3600.059	107	2912622	18.713	-0.4736
		10	2831758	3600.204	107	2836184	21.207	-0.1563

	20	1	3315613	1931.936	109	3325963	11.293	-0.3122
		2	3197797	3600.402	107	3206388	7.491	-0.2687
		3	3047695	3600.056	107	3047775	11.265	-0.0026
		4	3013601	3600.108	107	3013678	9.099	-0.0026
		5	2734356	3600.219	107	2738251	15.292	-0.1424
		6	3225209	3601.399	107	3225259	8.67	-0.0016
		7	3612783	3600.251	107	3618654	6.323	-0.1625
		8	2980250	3600.358	107	2980334	16.858	-0.0028
		9	2809433	946.518	109	2818858	16.157	-0.3355
		10	3335616	3600.108	107	3335679	7.679	-0.0019
	30	1	3109027	3600.437	107	3117735	11.224	-0.2801
		2	2981640	3600.686	107	2981670	15.31	-0.0010
		3	3209625	3600.396	107	3214072	7.669	-0.1386
		4	3399695	3600.673	107	3399720	7.154	-0.0007
		5	3142202	3600.374	107	3147562	10.908	-0.1706
		6	3657362	3600.153	107	3659671	6.272	-0.0631
		7	3390498	3600.66	107	3395345	8.276	-0.1430
		8	3508036	3600.19	107	3512831	10.581	-0.1367
		9	3998077	3602.077	107	4001165	11.243	-0.0772
		10	3512039	3600.102	107	3512050	10.802	-0.0003
15	10	1	4378642	3602.338	107	4435579	36.83	-1.3003
		2	4385814	3600.029	107	4422996	25.391	-0.8478
		3	4601152	3600.758	107	4628569	10.634	-0.5959
		4	4370563	3600.071	107	4411006	80.763	-0.9253
		5	4529265	3600.041	107	4562612	90.743	-0.7363
		6	4268914	3603.951	107	4321129	90.73	-1.2231
		7	4058466	3600.113	107	4125871	90.749	-1.6608
		8	4315210	3600.117	107	4369817	73.362	-1.2655
		9	4142610	3600.06	107	4197352	80.9	-1.3214
		10	4028925	3600.065	107	4085959	70.631	-1.4156
	20	1	5221012	3601.945	107	5221403	11.827	-0.0075
		2	5098258	3600.363	107	5114107	16.981	-0.3109
		3	4467573	3600.097	107	4477901	20.941	-0.2312
		4	4372377	3600.102	107	4411756	32.646	-0.9006
		5	4450816	3600.152	107	4490047	29.735	-0.8814
		6	5135328	3600.362	107	5140943	11.556	-0.1093
		7	5344191	3600.51	107	5352137	16.739	-0.1487
		8	4340220	3600.54	107	4358015	20.441	-0.4100
		9	4243058	3600.08	107	4244785	42.662	-0.0407

		10	4950853	2861.469	109	4971909	27.983	-0.4253
	30	1	4607692	3600.119	107	4612962	11.289	-0.1144
		2	4217139	3600.067	107	4231845	32.626	-0.3487
		3	4801997	3600.361	107	4811305	18.862	-0.1938
		4	5344445	3600.098	107	5350587	12.201	-0.1149
		5	4979308	3600.087	107	4985615	22.358	-0.1267
		6	5543007	3602.225	107	5551034	13.461	-0.1448
		7	5035976	3600.153	107	5050284	14.581	-0.2841
		8	5016436	3600.258	107	5026460	20.526	-0.1998
		9	6162688	3601.272	107	6165733	12.876	-0.0494
		10	5101036	3600.14	107	5109194	21.942	-0.1599
20	10	1	4416449	3600.151	107	4456197	60.944	-0.9000
		2	4585953	3600.066	107	4628011	12.511	-0.9171
		3	4577207	3600.027	107	4628083	17.935	-1.1115
		4	4529096	3600.252	107	4571407	17.773	-0.9342
		5	4559336	3600.054	107	4598196	16.853	-0.8523
		6	4542628	3600.173	107	4572502	31.584	-0.6576
		7	4373480	3600.071	107	4421131	19.326	-1.0895
		8	4389049	3600.098	107	4421462	8.865	-0.7385
		9	4528171	3601.84	107	4554606	15.744	-0.5838
		10	4451577	3600.094	107	4538080	17.622	-1.9432
	20	1	6507212	3600.089	107	6607329	30.869	-1.5386
		2	6452135	3600.114	107	6540068	46.621	-1.3629
		3	5931490	3600.093	107	5990005	35.077	-0.9865
		4	5966321	3600.137	107	6060176	71.624	-1.5731
		5	5973965	3600.11	107	6012405	43.708	-0.6435
		6	7030884	3600.088	107	7127351	22.735	-1.3720
		7	6928840	3600.093	107	6967739	36.532	-0.5614
		8	5802242	3600.201	107	5816364	39.807	-0.2434
		9	5414062	3600.147	107	5500260	61.451	-1.5921
		10	6642370	3600.115	107	6692544	41.245	-0.7554
	30	1	6154801	3605.345	107	6198551	15.679	-0.7108
		2	5904740	3600.109	107	5968860	45.683	-1.0859
		3	6358436	3600.109	107	6425674	135.609	-1.0575
		4	7213758	3600.119	107	7282004	16.292	-0.9461
		5	6470038	3600.182	107	6499874	24.672	-0.4611
		6	7263879	3600.152	107	7283296	24.654	-0.2673
		7	6404305	3600.116	107	6466879	34.249	-0.9771
		8	6653243	3604.087	107	6687620	28.699	-0.5167

		9	7805114	3600.236	107	7850004	11.936	-0.5751
		10	6564150	3600.136	107	6601791	24.911	-0.5734

Detailed computational results for MKPS

Table A.4: Detailed computational results for **MKPS** with $n_i \in [40,60]$

T	N	Instance	IP_{OBJ}	IP_{CPU}	IP_{stat}	VND&IP_{OBJ}	VND&IP_{CPU}	Gap (%)
5	10	1	480360	5,266	101	480360	1,327	0,0000
		2	361474	858,505	101	361474	0,899	0,0000
		3	388363	3600,106	107	388367	2,384	-0,0010
		4	117887	3600,057	107	117889	0,513	-0,0017
		5	356586	21,148	101	356586	0,964	0,0000
		6	148026	3600,032	107	148027	1,803	-0,0007
		7	500642	3600,044	107	500642	1,956	0,0000
		8	325009	3600,084	107	325010	0,776	-0,0003
		9	253131	3600,034	107	253131	1,464	0,0000
		10	439178	44,065	101	439178	2,109	0,0000
	20	1	207364	808,592	101	207364	2,499	0,0000
		2	349637	45,142	101	349637	2,143	0,0000
		3	193073	3600,002	107	193073	1,182	0,0000
		4	127765	3600,012	107	127765	1,287	0,0000
		5	529374	3600,048	107	529374	0,585	0,0000
		6	315280	3600,023	107	315280	2,064	0,0000
		7	207154	3600,011	107	207154	1,039	0,0000
		8	268123	3600,041	107	268123	2,009	0,0000
		9	257988	3600,071	107	257988	2,242	0,0000
		10	128720	3600,053	107	128720	0,63	0,0000
	30	1	509866	12,939	101	509866	2,009	0,0000
		2	274124	3600,128	107	274126	0,567	-0,0007
		3	568422	3600,075	107	568425	1,703	-0,0005

		4	396059	3600,038	107	396059	0,907	0,0000
		5	335804	1877,045	101	335804	3,193	0,0000
		6	374708	876,93	101	374708	1,964	0,0000
		7	307477	3600,096	107	307477	0,511	0,0000
		8	593428	96,971	101	593428	1,929	0,0000
		9	379861	22,729	101	379861	1,361	0,0000
		10	472679	23,819	101	472679	1,235	0,0000
10	10	1	610093	3600,031	107	610117	2,25	-0,0039
		2	496792	1691,307	101	496792	2,207	0,0000
		3	875876	1682,999	101	875876	3,575	0,0000
		4	221027	3600,035	107	221030	3,58	-0,0014
		5	796061	3600,087	107	796078	4,305	-0,0021
		6	291026	3600,021	107	291027	1,724	-0,0003
		7	917444	3600,051	107	917454	3,424	-0,0011
		8	494466	3600,063	107	494471	3,954	-0,0010
		9	363556	3600,04	107	363562	0,787	-0,0017
		10	563584	3462,333	109	563597	2,785	-0,0023
	20	1	339599	3600,104	107	339608	3,34	-0,0027
		2	598401	3600,047	107	598403	2,521	-0,0003
		3	351513	57,041	101	351513	3,995	0,0000
		4	321589	3600,11	107	321599	1,036	-0,0031
		5	715842	159,28	101	715842	8,737	0,0000
		6	512131	3600,089	107	512148	9,019	-0,0033
		7	537706	3600,028	107	537712	8,136	-0,0011
		8	499095	3600,05	107	499101	4,489	-0,0012
		9	423118	3600,035	107	423119	2,322	-0,0002
		10	279981	3600,008	107	279985	2,672	-0,0014
	30	1	814424	3600,142	107	814428	2,431	-0,0005
		2	588078	3600,052	107	588095	2,654	-0,0029
		3	897381	3600,017	107	897384	4,992	-0,0003
		4	603216	3600,146	107	603221	2,469	-0,0008
		5	681399	3600,013	107	681405	3,758	-0,0009
		6	670817	3600,052	107	670827	3,462	-0,0015
		7	594029	3600,167	107	594031	4,61	-0,0003
		8	896409	3600,17	107	896416	2,803	-0,0008
		9	792974	3600,053	107	792979	2,748	-0,0006
		10	620144	3600,095	107	620145	2,828	-0,0002
15	10	1	819641	2730,506	109	819659	2,481	-0,0022
		2	577960	3599,999	107	577963	3,617	-0,0005

		3	884048	3600,098	107	884050	4,171	-0,0002
		4	344198	3600,017	107	344198	1,035	0,0000
		5	1026123	806,8	109	1026154	4,015	-0,0030
		6	461658	3600,049	107	461661	11,822	-0,0006
		7	997702	964,571	109	997759	2,945	-0,0057
		8	620290	3572,608	101	620290	8,998	0,0000
		9	502535	3600,056	107	502538	2,23	-0,0006
		10	677626	3600,039	107	677633	4,537	-0,0010
	20	1	553830	1974,576	109	553840	3,309	-0,0018
		2	760429	3600,056	107	760447	8,88	-0,0024
		3	588806	3600,047	107	588818	3,441	-0,0020
		4	519449	3600,021	107	519449	4,294	0,0000
		5	851364	2342,401	109	851400	3,703	-0,0042
		6	675207	3600,062	107	675184	4,46	0,0034
		7	737089	2578,603	109	737113	3,087	-0,0033
		8	701381	3600,146	107	701419	17,694	-0,0054
		9	645277	3600,102	107	645284	11,039	-0,0011
		10	552512	3600,058	107	552529	6,673	-0,0031
	30	1	1260569	3600,377	107	1260604	6,464	-0,0028
		2	885019	3600,063	107	885047	7,354	-0,0032
		3	1261351	3600,078	107	1261374	6,005	-0,0018
		4	673490	3600,065	107	673491	3,82	-0,0001
		5	1106865	3600,399	109	1106890	5,289	-0,0023
		6	1030768	3600,114	107	1030782	14,371	-0,0014
		7	879557	3600,251	107	884538	14,779	-0,5663
		8	1191814	3601,261	107	1197606	7,502	-0,4860
		9	1226523	3600,061	107	1226524	4,722	-0,0001
		10	946300	3601,481	107	946316	5,56	-0,0017
20	10	1	923998	2588,191	109	924021	4,492	-0,0025
		2	675340	3600,022	107	675357	4,172	-0,0025
		3	918402	1554,004	109	918414	5,378	-0,0013
		4	393822	3600,098	107	393872	4,763	-0,0127
		5	1204986	3600,096	107	1204983	3,897	0,0002
		6	566384	3600,107	107	566400	6,208	-0,0028
		7	1119915	2506,488	109	1119926	3,905	-0,0010
		8	725767	3600,018	107	725825	4,408	-0,0080
		9	608267	3600,115	107	608272	2,606	-0,0008
		10	992463	1661,649	109	992482	7,233	-0,0019
	20	1	553665	2500,815	109	553690	8,55	-0,0045

		2	956338	3600,052	107	956392	4,656	-0,0056
		3	637619	1146,198	109	637633	2,484	-0,0022
		4	588845	1879,276	109	588855	4,46	-0,0017
		5	975610	3600,101	107	975644	4,212	-0,0035
		6	930799	201,652	101	930799	5,881	0,0000
		7	935814	2250,553	109	935839	3,81	-0,0027
		8	795586	3600,173	107	798064	8,513	-0,3115
		9	740334	1419,068	109	740355	5,909	-0,0028
		10	625550	3600,08	107	625567	7,089	-0,0027
	30	1	1786488	2469,467	109	1788539	8,107	-0,1148
		2	1081535	3600,055	107	1081552	4,717	-0,0016
		3	1755270	3600,087	107	1755275	6,075	-0,0003
		4	992951	3600,208	107	993010	5,454	-0,0059
		5	1279696	3600,174	107	1279719	5,174	-0,0018
		6	1498240	3600,154	107	1498250	5,91	-0,0007
		7	998275	3600,277	107	998370	4,648	-0,0095
		8	1375688	3600,326	107	1382694	6,837	-0,5093
		9	1668465	3600,588	107	1668526	6,027	-0,0037
		10	1144094	3600,355	107	1146313	19,649	-0,1940

Table A.5: Detailed computational results for **MKPS** with $n_i \in [60,90]$

T	N	*Instance*	$\mathrm{IP}_{\mathrm{OBJ}}$	$\mathrm{IP}_{\mathrm{CPU}}$	$\mathrm{IP}_{\mathrm{stat}}$	$\mathrm{VND\&}IP_{\mathrm{OBJ}}$	$\mathrm{VND\&}IP_{\mathrm{CPU}}$	*Gap* (%)
5	10	1	749278	3600,01	107	749278	4,321	0,0000
		2	893899	3600,106	107	893899	1,006	0,0000
		3	467063	3600,057	107	467067	5,277	-0,0009
		4	918335	19,598	101	918335	2,961	0,0000
		5	568503	2605,309	109	568523	4,294	-0,0035
		6	714083	178,41	101	714083	1,972	0,0000
		7	948393	3600,136	107	948406	1,382	-0,0014
		8	772142	2382,193	109	772153	2,646	-0,0014
		9	470959	3501,893	109	470960	2,771	-0,0002
		10	1165714	8,527	101	1165714	2,435	0,0000
	20	1	777924	3504,906	109	777924	2,687	0,0000
		2	972858	2512,766	109	972864	2,324	-0,0006
		3	1007536	3600,036	107	1007536	2,615	0,0000
		4	596031	3600,054	107	596031	4,554	0,0000
		5	917573	7,636	101	917573	2,076	0,0000
		6	1064355	3600,073	107	1064355	3,225	0,0000

		7	494558	23,163	101	494558	3,907	0,0000
		8	1249429	3600,077	107	1249428	1,517	0,0001
		9	887281	39,399	101	887281	8,604	0,0000
		10	743501	20,757	101	743501	5,384	0,0000
	30	1	638862	3600,017	107	638862	8,974	0,0000
		2	811968	70,597	101	811968	24,233	0,0000
		3	863958	394,231	101	863958	27,572	0,0000
		4	673064	3600,075	107	673064	28,113	0,0000
		5	836236	3600,139	107	836237	20,74	-0,0001
		6	933965	3600,047	107	933975	15,886	-0,0011
		7	950905	1373,417	109	950906	10,061	-0,0001
		8	789200	111,493	101	789200	18,053	0,0000
		9	599929	3600,212	107	599930	9,116	-0,0002
		10	985048	1086,568	101	985048	6,904	0,0000
10	10	1	1297180	3600,147	107	1297203	19,383	-0,0018
		2	1748379	1653,586	109	1748403	7,81	-0,0014
		3	819951	3600,071	107	819979	14,529	-0,0034
		4	1898854	2722,645	109	1899055	19,494	-0,0106
		5	1355062	1832,642	109	1355075	17,231	-0,0010
		6	1090736	3171,121	109	1090759	12,462	-0,0021
		7	1854791	2210,89	109	1854817	17,483	-0,0014
		8	1668233	3014,158	109	1668238	20,91	-0,0003
		9	1079277	1642,028	109	1079300	10,908	-0,0021
		10	1990918	1521,293	109	1991190	10,718	-0,0137
	20	1	1339693	3600,231	107	1339708	7,245	-0,0011
		2	1796794	2792,738	109	1796800	6,975	-0,0003
		3	2101876	2422,095	109	2101888	9,17	-0,0006
		4	1190683	2520,438	109	1190705	23,956	-0,0018
		5	1806710	3600,05	107	1806717	14,593	-0,0004
		6	1763092	3600,036	107	1763098	12,122	-0,0003
		7	1216524	2363,98	109	1216538	11,627	-0,0012
		8	2516774	3600,04	107	2516778	9,656	-0,0002
		9	1937644	3600,158	107	1937650	15,721	-0,0003
		10	1467312	3600,093	107	1467320	13,829	-0,0005
	30	1	1589088	2303,228	109	1589119	18,758	-0,0020
		2	1589561	3600,118	107	1589577	9,326	-0,0010
		3	1576024	3600,102	107	1576024	10,931	0,0000
		4	1401505	3601,053	107	1401567	7,329	-0,0044
		5	1549640	3600,294	107	1549652	9,185	-0,0008

		6	1788069	3600,165	107	1788074	12,329	-0,0003
		7	1562270	3600,118	107	1562284	9,403	-0,0009
		8	1506371	3600,279	107	1506388	14,015	-0,0011
		9	1194680	3600,345	107	1194692	14,466	-0,0010
		10	1755413	3600,281	107	1759679	11,281	-0,2430
15	10	1	1739689	2142,132	109	1739769	22,211	-0,0046
		2	1970440	1748,074	109	1970442	9,563	-0,0001
		3	1106748	3600,06	107	1106752	10,87	-0,0004
		4	2191918	1125,101	109	2191926	9,135	-0,0004
		5	1581821	1838,908	109	1581844	33,013	-0,0015
		6	1560054	77,669	101	1560054	6,199	0,0000
		7	2333380	3600,104	107	2333382	6,775	-0,0001
		8	2023994	963,411	109	2024007	9,765	-0,0006
		9	1253974	3112,038	109	1254002	6,833	-0,0022
		10	2421756	1505,461	109	2421781	5,824	-0,0010
	20	1	1881235	3600,503	107	1881258	13,003	-0,0012
		2	2674926	3600,291	107	2674973	13,19	-0,0018
		3	3021558	2539,329	109	3021582	16,454	-0,0008
		4	1627038	3600,304	107	1627329	8,242	-0,0179
		5	2497873	2998,675	109	2497922	11,059	-0,0020
		6	2476359	2985,436	109	2476414	10,294	-0,0022
		7	1928438	3600,25	107	1928454	9,068	-0,0008
		8	3562412	3600,433	107	3562434	11,172	-0,0006
		9	2694051	3600,303	107	2699463	9,793	-0,2009
		10	2192372	3600,246	107	2196997	18,965	-0,2110
	30	1	2154158	3604,183	107	2154204	12,868	-0,0021
		2	2482826	3600,237	107	2482862	16,546	-0,0014
		3	2381005	3608,127	107	2381037	13,667	-0,0013
		4	2336182	3600,729	107	2340333	43,878	-0,1777
		5	2198978	3602,512	107	2199001	16,146	-0,0010
		6	2607729	3603,155	107	2607751	20,436	-0,0008
		7	2266615	3601,136	107	2266647	19,477	-0,0014
		8	2326888	3601,31	107	2326934	18,665	-0,0020
		9	1666442	3602,47	107	1666455	17,883	-0,0008
		10	2596631	3601,64	107	2596684	21,938	-0,0020
20	10	1	1892823	1328,582	109	1892865	12,149	-0,0022
		2	2095319	3600,378	107	2095325	6,345	-0,0003
		3	1324191	3600,119	107	1324206	8,444	-0,0011
		4	2287623	2077,323	109	2287639	8,636	-0,0007

		5	1730296	1891,115	109	1730323	10,313	-0,0016
		6	1649880	3600,107	107	1649894	7,371	-0,0008
		7	2313764	1623,323	109	2313807	8,224	-0,0019
		8	2195281	1097,596	109	2195296	9,481	-0,0007
		9	1356874	2165,755	109	1356889	7,512	-0,0011
		10	2612314	3662,126	107	2612325	4,34	-0,0004
	20	1	2054664	3600,253	107	2059531	12,284	-0,2369
		2	3414528	3600,349	107	3415761	22,812	-0,0361
		3	3346392	3600,207	107	3346427	21,893	-0,0010
		4	2181427	3600,343	107	2181569	19,396	-0,0065
		5	3175307	3600,321	107	3182993	22,222	-0,2421
		6	3357872	2029,553	109	3358128	26,696	-0,0076
		7	2508798	3600,29	107	2514565	45,745	-0,2299
		8	4504937	3600,478	107	4507022	18,158	-0,0463
		9	3293329	3600,133	107	3293609	18,641	-0,0085
		10	2680841	3600,35	107	2697052	20,349	-0,6047
	30	1	2878317	3601,075	107	2887193	39,929	-0,3084
		2	3191596	3600,855	107	3198538	31,801	-0,2175
		3	2888584	3337,014	109	2888636	18,224	-0,0018
		4	2983602	3600,919	107	2994244	21,968	-0,3567
		5	2898612	3600,833	107	2901696	15,055	-0,1064
		6	3492396	3601,721	107	3492511	14,856	-0,0033
		7	2742243	3600,762	107	2742348	17,872	-0,0038
		8	2862295	3601,287	107	2863988	21,09	-0,0591
		9	2137698	3601,234	107	2140943	18,427	-0,1518
		10	3108535	3600,587	107	3112329	22,761	-0,1221

Table A.6: Detailed computational results for **MKPS** with $n_i \in [90,110]$

T	N	$Instance$	IP_{OBJ}	IP_{CPU}	IP_{stat}	$VND\&IP_{OBJ}$	$VND\&IP_{CPU}$	$Gap(\%)$
5	10	1	656534	915,108	101	656534	8,131	0,0000
		2	423931	3600,023	107	423931	4,495	0,0000
		3	494401	297,705	101	494401	10,513	0,0000
		4	365790	3600,119	107	365790	6,129	0,0000
		5	874728	3600,061	107	874728	6,529	0,0000
		6	654887	3600,017	107	654889	10,18	-0,0003
		7	761701	3600,056	107	761702	4,161	-0,0001
		8	695677	3600,067	107	695677	2,957	0,0000

		9	197419	243,39	101	197419	4,956	0,0000
		10	376848	1732,863	101	376848	1,509	0,0000
	20	1	293214	3600,304	107	293214	5,701	0,0000
		2	510146	3600,104	107	510149	2,729	-0,0006
		3	529011	3600,023	107	529014	7,53	-0,0006
		4	373967	3600,047	107	373967	4,686	0,0000
		5	733780	3600,128	107	733780	3,876	0,0000
		6	582963	3600,067	107	582963	6,328	0,0000
		7	593663	3600,135	107	593663	1,73	0,0000
		8	412685	3600,07	107	412686	6,017	-0,0002
		9	401871	3600,083	107	401871	5,292	0,0000
		10	501253	13,489	101	501253	14,463	0,0000
	30	1	354479	3589,983	107	354479	14,613	0,0000
		2	402799	3600,093	107	402799	14,682	0,0000
		3	575269	3600,088	107	575269	12,866	0,0000
		4	244253	3600,254	107	244253	18,19	0,0000
		5	722704	52,977	101	722704	11,762	0,0000
		6	812867	3600,076	107	812868	17,463	-0,0001
		7	546043	404,245	101	546043	14,032	0,0000
		8	398888	3600,029	107	398890	13,585	-0,0005
		9	908159	1811,026	101	908159	25,871	0,0000
		10	625340	3600,247	107	625341	22,318	-0,0002
10	10	1	980287	3447,886	109	980296	9,245	-0,0009
		2	950203	3125,25	109	950210	2,737	-0,0007
		3	1324721	3600,117	107	1324742	4,769	-0,0016
		4	565865	2437,203	109	565883	6,576	-0,0032
		5	1536421	1193,358	109	1536428	9,462	-0,0005
		6	1186472	3600,031	107	1186475	8,032	-0,0003
		7	1147656	3600,098	107	1147679	7,255	-0,0020
		8	1043693	3600,071	107	1043696	6,459	-0,0003
		9	512428	1208,431	109	512433	2,862	-0,0010
		10	1042498	3600,101	107	1042498	8,133	0,0000
	20	1	566741	1405,945	109	566746	1,689	-0,0009
		2	751168	2447,704	109	751174	4,156	-0,0008
		3	1138940	3600,027	107	1138952	5,804	-0,0011
		4	687320	3600,143	107	687321	3,374	-0,0001
		5	1055685	3600,188	107	1055692	6,2	-0,0007
		6	1014574	2362,184	109	1014583	5,869	-0,0009
		7	1263824	3600,171	107	1263827	6,065	-0,0002

		8	977119	3600,236	107	977121	4,351	-0,0002
		9	759552	3600,154	107	759585	5,657	-0,0043
		10	1170952	3322,323	109	1170960	4,324	-0,0007
	30	1	924062	3600,345	107	926132	14,805	-0,2240
		2	795307	3600,189	107	795308	24,456	-0,0001
		3	1151123	3600,369	107	1151135	15,736	-0,0010
		4	1054426	3600,36	107	1055163	13,303	-0,0699
		5	1001320	3600,297	107	1001631	17,017	-0,0311
		6	1528497	3600,368	107	1528523	13,969	-0,0017
		7	892826	3600,444	107	892830	21,839	-0,0004
		8	863045	3600,282	107	863321	28,975	-0,0320
		9	1601469	2700,861	109	1601491	42,246	-0,0014
		10	1052427	3600,245	107	1052889	19,927	-0,0439
15	10	1	1299020	3029,444	109	1299026	12,989	-0,0005
		2	1370884	3600,094	107	1370892	16,941	-0,0006
		3	1753986	3600,135	107	1753995	11,213	-0,0005
		4	658266	3203,556	109	658276	11,423	-0,0015
		5	1864064	3357,008	109	1864083	12,938	-0,0010
		6	1480934	3600,098	107	1481002	17,347	-0,0046
		7	1590736	1451,957	109	1590718	12,279	0,0011
		8	1625117	2395,634	109	1625154	13,372	-0,0023
		9	821419	2711,682	109	821434	26,963	-0,0018
		10	1238994	1877,664	109	1239007	27,571	-0,0010
	20	1	751496	3600,146	107	751503	8,784	-0,0009
		2	1236409	3600,345	107	1236434	11,094	-0,0020
		3	1346215	2412,845	109	1346230	28,768	-0,0011
		4	1111093	3600,275	107	1111108	11,478	-0,0014
		5	1444617	3600,221	107	1444629	12,854	-0,0008
		6	1521224	3600,096	107	1521649	14,308	-0,0279
		7	1483334	3600,065	107	1483343	15,772	-0,0006
		8	1010548	2213,352	109	1010559	29,272	-0,0011
		9	1184838	3200,088	109	1184844	11,112	-0,0005
		10	1415246	3600,283	107	1420656	14,563	-0,3823
	30	1	1143214	3600,508	107	1143248	12,275	-0,0030
		2	1203139	2869,664	109	1203156	10,509	-0,0014
		3	1538637	3601,674	107	1540040	33,465	-0,0912
		4	1556610	3601,249	107	1556658	11,946	-0,0031
		5	1675140	3603,108	107	1677523	11,135	-0,1423
		6	2324162	3606,788	107	2324174	22,627	-0,0005

		7	1508000	3600,983	107	1513957	16,22	-0,3950
		8	1155112	3601,642	107	1155148	30,225	-0,0031
		9	2544976	3601,508	107	2570083	11,048	-0,9865
		10	1447433	2748,247	109	1452778	15,932	-0,3693
20	10	1	1529304	3600,195	107	1529329	12,156	-0,0016
		2	1604158	1299,673	109	1604175	10,366	-0,0011
		3	1851066	3600,026	107	1851068	13,443	-0,0001
		4	868624	2645,082	109	868623	28,927	0,0001
		5	2197968	1553,864	109	2197982	13,697	-0,0006
		6	1760881	3600,083	107	1760895	14,157	-0,0008
		7	1852092	3600,444	107	1852101	12,145	-0,0005
		8	2030304	3600,115	107	2030308	18,675	-0,0002
		9	1000100	1892,344	109	1000245	18,07	-0,0145
		10	1468818	2811,26	109	1468848	12,267	-0,0020
	20	1	1054158	3600,614	107	1070162	19,861	-1,5182
		2	1597433	2541,067	109	1597454	10,794	-0,0013
		3	1941353	3607,082	107	1941365	18,487	-0,0006
		4	1363260	3600,452	107	1363284	13,046	-0,0018
		5	1559026	3600,597	107	1564472	13,741	-0,3493
		6	1820157	2038,711	109	1820179	48,714	-0,0012
		7	2225146	3600,552	107	2245598	15,039	-0,9191
		8	1517157	3600,482	107	1519930	17,561	-0,1828
		9	1722099	3600,769	107	1722162	15,164	-0,0037
		10	2010704	3600,414	107	2024897	18,286	-0,7059
	30	1	1617864	3600,181	107	1617890	28,243	-0,0016
		2	1616378	3600,833	107	1616459	46,824	-0,0050
		3	1999500	3600,767	107	2018955	39,578	-0,9730
		4	2215157	3600,859	107	2225528	35,995	-0,4682
		5	2159001	3603,322	107	2159032	29,901	-0,0014
		6	2955832	3600,201	107	2955977	35,069	-0,0049
		7	1791356	3600,843	107	1810016	20,996	-1,0417
		8	1748618	3601,199	107	1750200	20,857	-0,0905
		9	3341939	3601,423	107	3373311	13,412	-0,9387
		10	1939662	3611,679	107	1939708	29,37	-0,0024

Detailed computational results for MCKS

Table A.7: Detailed computational results for **MCKS**

T	N	$Instance$	IP_{OBJ}	IP_{CPU}	IP_{stat}	$VNS\&IP_{OBJ}$	$VNS\&IP_{CPU}$	$Gap(\%)$
5	10	1	1752164	11,089	101	1752164	0,609	0,0000
		2	1840553	7,29	101	1840553	0,562	0,0000
		3	1781144	3578,437	109	1781144	0,577	0,0000
		4	1791770	3600,054	107	1791778	1,56	-0,0004
		5	1777694	789,434	101	1777694	1,435	0,0000
		6	1766853	2112,405	109	1778441	1,139	-0,6559
		7	1736342	3600,034	107	1736345	1,342	-0,0002
		8	1708739	3600,005	107	1708739	1,17	0,0000
		9	1798198	42,86	101	1798198	0,67	0,0000
		10	1769033	6,04	101	1769033	0,452	0,0000
	20	1	3529051	3600,195	107	3529051	6,739	0,0000
		2	3574108	498,482	101	3574108	4,088	0,0000
		3	3609997	2681,087	109	3610006	8,892	-0,0002
		4	3572567	2790,697	109	3572580	14,878	-0,0004
		5	3560930	2965,372	109	3574932	13,447	-0,3932
		6	3628358	2952,339	109	3628358	14,789	0,0000
		7	3610827	3600,135	107	3610835	15,726	-0,0002
		8	3444725	2981,399	109	3452736	14,198	-0,2326
		9	3497325	3094,946	109	3497327	15,054	-0,0001
		10	3687252	3466,024	101	3687252	5,148	0,0000
	30	1	5420618	369,414	101	5420618	20,015	0,0000
		2	5341566	907,866	101	5341566	29,656	0,0000
		3	5415437	3600,05	107	5415434	24,352	0,0001
		4	5504079	861,525	101	5504079	21,103	0,0000
		5	5407728	3600,23	107	5407725	25,479	0,0001
		6	5389200	3600,201	107	5400757	30,496	-0,2144
		7	5379790	3600,109	107	5381312	29,135	-0,0283
		8	5304058	2049,279	101	5301471	20,076	0,0488
		9	5305506	3600,286	107	5321473	27,66	-0,3010
		10	5516312	478,553	101	5509704	19,968	0,1198
10	10	1	1774956	3600,181	107	1774956	1,037	0,0000
		2	1793067	3600,112	107	1793067	0,986	0,0000
		3	1833041	24,571	101	1833041	1,522	0,0000

		4	1813713	3600,029	107	1813714	2,481	-0,0001
		5	1795947	2176,963	109	1795947	1,281	0,0000
		6	1812125	2551,292	109	1812125	1,801	0,0000
		7	1741272	2550,081	109	1741272	1,328	0,0000
		8	1735084	2993,187	109	1735085	1,985	-0,0001
		9	1833775	3248,65	101	1833775	1,87	0,0000
		10	1818894	1520,797	101	1818894	0,799	0,0000
	20	1	3579731	3600,219	107	3579733	16,341	-0,0001
		2	3581346	3600,214	107	3583358	15,796	-0,0562
		3	3597838	2842,868	109	3597872	16,459	-0,0009
		4	3577677	3600,791	107	3577745	16,329	-0,0019
		5	3667813	3600,133	107	3667814	11,373	0,0000
		6	3645837	3600,228	107	3645837	16,382	0,0000
		7	3682509	2741,637	101	3678369	7,631	0,1124
		8	3486724	3600,211	107	3486742	15,56	-0,0005
		9	3583027	3600,203	107	3579285	16,794	0,1044
		10	3627056	3600,218	107	3629775	16,77	-0,0750
	30	1	5404545	3600,394	107	5404545	32,701	0,0000
		2	5410704	2913,27	101	5410704	28,537	0,0000
		3	5397028	2509,967	101	5397028	30,136	0,0000
		4	5586034	3600,347	107	5586034	33,533	0,0000
		5	5471288	2300,482	101	5471288	35,676	0,0000
		6	5514079	3600,163	107	5514079	33,388	0,0000
		7	5469693	43,879	101	5469693	26,183	0,0000
		8	5345750	3600,254	107	5345751	27,89	0,0000
		9	5347158	3600,572	107	5353704	38,26	-0,1224
		10	5504324	3600,148	107	5504324	34,714	0,0000
15	10	1	1744422	3600,188	107	1759526	11,51	-0,8658
		2	1816781	52,36	101	1816781	2,67	0,0000
		3	1833898	3600,048	107	1833899	2,17	-0,0001
		4	1817425	3600,22	107	1817425	2,379	0,0000
		5	1801379	3181,59	109	1801379	3,456	0,0000
		6	1814556	3600,501	107	1814558	9,818	-0,0001
		7	1780651	3565,163	109	1780651	1,95	0,0000
		8	1743182	2440,431	109	1746953	2,817	-0,2163
		9	1812751	3600,251	107	1812751	6,814	0,0000
		10	1767045	945,889	109	1768693	2,731	-0,0933
	20	1	3556794	3600,404	107	3556844	17,911	-0,0014
		2	3603525	3600,284	107	3630413	18,224	-0,7462

		3	3605302	3600,21	107	3630366	17,245	-0,6952
		4	3637303	3600,445	107	3637340	20,453	-0,0010
		5	3594851	3600,117	107	3603616	20,839	-0,2438
		6	3684172	3600,638	107	3684179	18,902	-0,0002
		7	3619591	3600,699	107	3628287	17,92	-0,2402
		8	3448577	3596,866	107	3472721	17,116	-0,7001
		9	3598426	925,246	109	3612918	19,805	-0,4027
		10	3709433	3600,486	107	3713767	18,216	-0,1168
	30	1	5470811	3600,407	107	5470808	41,629	0,0001
		2	5392454	3600,376	107	5392484	40,022	-0,0006
		3	5453503	3600,293	107	5455858	36,365	-0,0432
		4	5597147	144,643	101	5597147	30,06	0,0000
		5	5502609	3600,292	107	5502609	36,784	0,0000
		6	5517831	3600,665	107	5519366	33,292	-0,0278
		7	5471749	3600,656	107	5483645	37,098	-0,2174
		8	5291390	3600,092	107	5336834	36,637	-0,8588
		9	5451912	3600,064	107	5451912	34,044	0,0000
		10	5563697	3600,474	107	5569464	41,036	-0,1037
20	10	1	1767421	3375,366	109	1767421	3,005	0,0000
		2	1842297	3600,076	107	1842310	3,599	-0,0007
		3	1779168	3600,251	107	1779168	2,104	0,0000
		4	1841122	31,027	101	1841122	1,74	0,0000
		5	1763882	3600,829	107	1797818	11,319	-1,9239
		6	1807989	3384,961	109	1807989	12,22	0,0000
		7	1766339	3600,251	107	1766345	9,95	-0,0003
		8	1755457	45,957	101	1755457	2,451	0,0000
		9	1807689	3230,208	109	1810502	2,844	-0,1556
		10	1799548	2978,951	109	1799552	5,372	-0,0002
	20	1	3611595	2405,53	101	3611595	15,374	0,0000
		2	3576350	3600,087	107	3601021	20,847	-0,6898
		3	3553751	3600,18	107	3556291	21,951	-0,0715
		4	3691772	3600,402	107	3692358	21,454	-0,0159
		5	3670597	3600,088	107	3670765	20,557	-0,0046
		6	3692909	3600,063	107	3692920	21,605	-0,0003
		7	3603229	3600,12	107	3615152	21,808	-0,3309
		8	3466735	3600,135	107	3487047	18,833	-0,5859
		9	3515371	3600,13	107	3519061	19,616	-0,1050
		10	3708739	3600,117	107	3708758	19,66	-0,0005
	30	1	5460699	3600,445	107	5463814	44,567	-0,0570

		2	5466949	3600,512	107	5469767	43,854	-0,0515
		3	5398644	3600,194	107	5418735	47,719	-0,3721
		4	5554985	80,056	101	5554985	42,032	0,0000
		5	5475670	3600,256	107	5476227	41,721	-0,0102
		6	5466058	3600,082	107	5486334	41,596	-0,3709
		7	5471791	3600,105	107	5483799	42,6	-0,2195
		8	5320146	3600,101	107	5345186	41,576	-0,4707
		9	5392943	3600,414	107	5392943	44,002	0,0000
		10	5538874	723,181	101	5538874	42,515	0,0000