UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Dottorato di Ricerca in
Automatica e Ricerca Operativa

XVI Ciclo

Metaheuristic Algorithms for Combinatorial Optimization Problems

Manuel Iori

Il Coordinatore
Prof. Alberto Tonielli

Il Tutor
Prof. Silvano Martello

A.A. 2000–2003

Contents

Acknowledgments

Many thanks go to my tutor, Prof. Silvano Martello, for introducing me to the Operations Research and Combinatorial Optimization, and for helping me with important suggestions and encouragements that led me to the realization of this Ph.D. thesis.

I would like to thank all the components of the group of Operations Research of the DEIS, *Dipartimento di Elettronica, Informatica e Sistemistica* of the University of Bologna, namely Prof. Paolo Toth, Prof. Daniele Vigo, Prof. Alberto Caprara, Dr. Andrea Lodi, Dr. Michele Monaci and Ph.D. students Valentina Cacchiani and Enrico Malaguti, for helping me in the resolution of the problems that arise from addressing the Combinatorial Optimization field, and for being always kind in giving an answer to my questions and doubts.

Many thanks go also to my co-authors, that helped me in writing the reports that form the greatest part of this thesis and that led me to present submissions to publications in international journals of the field: Prof. Silvano Martello, Prof. Daniele Vigo and Dr. Michele MOnaci, from the University of Bologna, Prof. Mauro Dell'Amico from the University of Modena and Reggio Emilia, Prof. Eleni Hadjiconstantinou from the Imperial College, London, Prof. Juan José Salazar González, from the University of La laguna, Tenerife and Prof. Michel Gendreau and Prof. Gilbert Laporte, from the University of Montreal.

Many thanks go also to the director of my Ph.D. course, Prof. Alberto Tonielli, to the director of the DEIS, Prof. Claudio Bonivento and to the other Ph.D. students and people of the DEIS, for being always nice and friendly during these years.

I would like also to thank the "Ministero dell'Istruzione, dell'Università e della Ricerca" (MIUR) and the "Consiglio Nazionale delle Ricerche" (CNR), Italy, for the financial support given to this project.

Finally, I would like to thank my family for always helping me during these years of work at the University of Bologna.

Bologna, March 12, 2004

Manuel Iori

List of Figures

List of Tables

Preface

Combinatorial Optimization is a very interesting and challenging field of study. In the last 60 years many works have been published in which several theoretical and practical optimization problems have been addressed with various approaches.

Exact techniques have been applied in order to find optimal solutions to the given problems. Because of the complexity of the great part of the combinatorial problems, exacts are not able to prove optimality for "big-size" instances (where the concept of big-size is strictly dependent from the nature of the problems addressed). Moreover, the theory of complexity shows us how it is unlike that a polynomial time algorithm will ever be developed for NP-Hard problems.

In order to provide good feasible solutions for those cases in which exacts fail to reach the optimality, or simply for improving the performances of the exacts, heuristic techniques have been developed. They consist in approximation approaches, obtained by applying a tailored strategy to the input problem, according to some given criteria, and leading to a feasible output solution. They are generally fast, but their behavior is strictly dependent from the instance addressed and can change consistently according to changes in the input.

For the improvement of the heuristic approaches, techniques of local search have been used. They perform a search of the solutions that are "close" to a given input solution. The set of solutions explored is defined the neighborhood of the input. Local search techniques are able to reach a local optimum of the area explored, but, since they do not have a complete vision of the search space, fail in moving towards other, maybe better, local optima.

To avoid this limit of local search techniques, in the recent years metaheuristic algorithms have been created. They consist in performing intelligent search of the space of solutions, starting with one or more input solutions, and improving them through intense search in the promising areas and diversification for moving towards appealing areas. Many different techniques have been applied, leading to interesting results for many Combinatorial Optimization problems.

In this Ph.D. thesis we address relevant optimization problems, using exact, heuristic and local search techniques and especially focusing on metaheuristics. In Chapter 1, a brief introduction to the different techniques that can be developed is given. In the following chapters applications of these techniques are presented for the problems addressed. In Chapters 2 and 3, we investigate the area of Scheduling. In Chapters 4 and 5 we propose solution approaches to problems of multi-dimensional Cutting and Packing. Finally in Chapters 6 and 7 we investigate the area of Transportation, presenting a new Vehicle Routing Problem with multi-dimensional constraints on the vehicles.

Part I

Introduction

Chapter 1

On the MetaHeuristic Algorithms

Despite the quite recent introduction of metaheuristic techniques in the Operations Research field, there is a huge number of publications reporting successful results of these approaches for NP-Hard problems. The interest in these techniques remains particularly vivid in these years for many motivations: the high flexibility, that leads also to the possibility of re-use of the software, and the good heuristic performances, that allow to address efficiently big-size instances of difficult problems.

Many techniques are available, and researchers from all over the world keep on suggesting new ones. Some approaches, as, e.g., Tabu Search and Simulated Annealing, are based on iterated moves, that starting from an initial solution and according to certain criteria, move towards new solutions. The concept of *neighborhood*, i.e., the set of solutions that are "close" to a given solution, is central for these approaches. Other techniques, as, e.g., Genetic Algorithms and Scatter Search, are based on a population of solutions. The search process is performed by selecting subsets of this population and combining them in a tailored way, in order to obtain new solutions.

It is not clear to which extent is important to choose one technique instead of another for efficiently addressing NP-Hard problems. There is not a single technique that is leading, with respect to the computational results obtained, over the others, and different techniques showed to produce similar results when referred to the same problem. It is clear instead how the use of information coming from the knowledge of the specific problem can help consistently the approach (as, e.g., for the hybridization in the Genetic Algorithms). It is also clear that, in the metaheuristic approach used, one has to find a good compromise between the necessity of exploring intensely the most promising areas and of diversifying the search process in order to move towards new areas of the search space.

These two aspects of the search approach are commonly defined, in metaheuristic literature, as *intensification* and *diversification* phase. Intensification is used when some criteria tell us that the area (or the solution) currently explored is of particular interest. It is usually implemented with Local Search techniques (as the frequently used $\lambda-$opt exchanges), but also with other metaheuristics, nested in the main approach and used with a limited computational time (as in the memetic algorithms). The diversification phase is used instead when we want to move the search from an area which is not considered appealing any more. It can be applied with regular frequency (as for mutation or immigration in the Genetic Algorithms) or only in some cases, when the approach has not been able to improve the incumbent solution in a certain number of iterations (as in the Multi-Start methods). The use of a long

term memory, keeping tracks of the history of the search process, proved to be an efficient technique for an accurate diversification.

In this chapter we briefly revise the main features of the metaheuristic approaches, focusing particularly on those used by the author in the following chapters of the Ph.D. thesis for addressing combinatorial optimization problems, namely: Scatter Search, Tabu Search and Genetic Algorithms.

In Section 1.1 we present Tabu Search techniques, focusing on the use of neighborhood structure and of tabu lists in order to perform a search process that, starting from an initial solution, moves in the search space without going back to its previous steps. In Section 1.2 we present Scatter Search and Path Relinking, two techniques proposed in the '60s, and rediscovered in recent years and applied efficiently to optimization problems. In Section 1.3 we present Genetic Algorithms, a technique that has a great impact on the Optimization field and that led to an enormous number of publications, giving details for some of its relevant features, as, e.g., crossover operators, mutation and evolution of a population of solutions.

In Section 1.4 we discuss the Simulated Annealing metaheuristic,based on an analogy to the process of physical annealing of solids. In Section 1.5 the Ant Colony Optimization is described, an approach that, despite some discouraging results, is used by a great community of researcher in optimization. Variable neighborhood search, a recent technique based on the change of neighborhood structure during the search process, is presented in Section 1.6.

For sake of conciseness, other metaheuristic techniques, as Multi-Start methods, Guided Local Search, Greedy Randomized Adaptive Search Procedure, Neural Networks and Memetic Algorithms, are only briefly described in Section 1.7.

For each technique a list of some successful approaches and of surveys is given. For a more detailed introduction to the field of metaheuristic, the interested reader could refer to, e.g., the recent book by Glover and Kochenberger [70].

1.1 Tabu search

Tabu Search (TS) was originally proposed by Glover in the 1980s [65], and since then has been successfully applied to several combinatorial optimization problems, generally providing solutions very close to optimality and constituting a good approach for handling difficult problems. These successes have made TS very popular in the Operations Research field, leading to hundreds of publications in the last years.

The basic principle of TS is to start from a heuristic solution, to improve it through Local Search (LS), and to pursue, whenever LS encounters a local optimum, by allowing non-improving moves. Cycling back to previously visited solutions is prevented by the use of memories, called *tabu lists*, that record the recent history of the search. It is of interest to note that, in the same year in which Glover proposed TS (1986), Hansen presented a similar approach, named *steepest ascent/ mildest descent*.

In the following sections we outline some key ideas of the approach.

1.1.1 Neighborhood Structure

The two main elements of any TS are the definitions of the *search space* and of the *neighborhood structure*. The search space of a LS or TS heuristic is simply the space of all the solutions that can be explored during the search. It should consist of all the feasible solutions (hence, the optimal ones), but could also contain infeasible solutions, if they are considered relevant

for the search. Indeed, it is not always a good idea to restrict the search space to feasible solutions, because in many cases allowing the search to move towards infeasible ones may be convenient, in order to achieve new better feasible values later on (see, e.g., Gendreau, Hertz and Laporte [58]).

Closely linked to the definition of search space is the one of neighborhood structure. At each iteration, the local transformation that can be applied to the current solution, denoted s, defines a set of neighboring solutions in the search space. This subset is defined the neighborhood of s ($N(s)$) and it consists in the subset of solutions that can be reached from s through a specific operations (or a specific set of operations). An operation that allows the algorithm to change a solution into another one is defined, in TS literature, a *move*. For any problem considered there are many neighborhood structures and they can be considerably different one from the other. Consequently there are many moving criteria for changing the actual solution. Choosing correctly both the search space and the neighborhood structure is by far the most critical step in the development of an effective TS approach.

1.1.2 Tabu Lists

Tabus are one of the most distinctive elements of TS. They consist of moves that are not allowed to be performed and are used in order to prevent cycling when leaving a local optimum through non-improving moves. The key realization here is that, when escaping from a minimum, something needs to be done in order to prevent the search from tracking back its steps to the same minimum where it came from. This is achieved by declaring tabu, (i.e., disallowing), moves that reverse the effect of other recent moves.

Tabus are sorted in a *short term memory* of the search, the so called *tabu list*. In this list only a fair amount of information is kept, both in terms of length of the list and of information for each tabu, in order not to waste too much computing time and memory. In any given context there are several possibilities regarding the specific information that is recorded, and finding a good compromise between memory storage and probability of not encountering any loop represents another key point for effective TS programming. The most commonly used tabus involve recording the last few transformations performed on the current solution and forbidding reverse transformations, others are based on key characteristics of the solutions themselves or of the moves. Because of these limits in the memory storage of tabus, it is generally not sure that a cycling will always be forbidden, and thus a certain level of randomization, or other diversification criteria are kept.

Standard tabu lists are usually implemented in a list of fixed length, but it has been shown that fixed length tabus cannot always prevent from cycling. An alternative is the use of a variable length for the tabu list, that can vary during the search process (see, e.g., Glover [66]).

While central to TS, tabus are sometimes too powerful, since they prohibit very attractive moves. This can also happen when there is no danger of cycling and the process is kept in the same area of the search space for a too long time. It is thus necessary to use algorithmic devices that will allow one to revoke tabus and to perform moves anyway. These are called *aspiration criteria*. The most natural criterion is to accept to perform a move, even if it is tabu, if it would lead to a new incumbent solution with a better objective value. Other aspiration criteria are based on similar considerations, or on the idea of aspiration level: if the move is tabu but would lead to a solution of a certain quality (level), the move is performed.

1.1.3 A Simple Template for Tabu Search

Suppose that we are trying to minimize an objective function $f(s)$, over some domain, and that we apply the best improvement verison of TS, i.e., the one in which the best move is computed at each iteration. Define s the current solution, $\overline{s}$ the incumbent solution, $f(\overline{s})$ the value of the objective corresponding to the incumbent solution, $N(s)$ the neighborhood, and $N^*(s)$ the admissible neighborhood (i.e., the set of solutions in $N(s)$ which are not tabu).

The initialization is obtained by constructing an initial solution s_0 and setting $s = s_0$, $f(\overline{s}) = f(s_0)$, $\overline{s} = s_0$ and $T = \emptyset$, where T is the set of tabu moves. The search is then operated by selecting in $N(s)$ the solution s' for which $f(s')$ is minimum, by eventually updating $\overline{s}$, and by updating T inserting the value of the last move. The search continues until some stopping criteria are met. Typically a maximum elapsed time, a maximum number of iterations, a proof of optimality for the incumbent solution (coming from previous information on the lower bound), or a maximum number of iterations without improvements in the incumbent solution.

1.1.4 Intensification and Diversification

Additional elements have to be included in the strategy in order to make it effective. The most important among those possible additional features are the *intensification* of the search, applied in those areas of the search space that seem to be particularly appealing, and the *diversification*, in order to escape from poor local minima and move towards more promising areas.

Concerning intensification, from time to time one should stop the normal search in order to perform a more tailored search in some region of the space. This can be done through several different techniques. Some common techniques are, e.g., changing the structure of the neighborhood and allowing the performing of different moves, or using some procedure tailored for a local search. Two-opt, and generally $\lambda-$ opt procedures, that try to exchange at the same time λ items among them and consequently improve the objective function, proved to be very useful in this context, and have been efficiently adapted to many combinatorial optimization problems.

If the searching process tends to be too restricted to a limited area of the search space, an algorithmic device is needed to force the search toward different areas. It is usually based on some form of long time memory of the search, such as frequency, in which one records the total number of iterations, since the beginning of the TS, in which a solution, or a solution feature has been selected. The diversification is then obtained in continuous form, by giving a chance at each iteration of moving from the area (as, e.g., later on in the Guided Local Search of Section 1.7.1), or in a restart form, that iteratively stops the TS and restarts it from a different point in the search space (as, e.g., later on for the Multi Start Methods in Section 1.7.2).

1.1.5 Some successful applications of Tabu Search

We refer to the recent survey of Gendreau [57], for a list of successful approaches of TS to combinatorial optimization problems. We give some more details for two aspects of combinatorial optimization for which many TS approaches were developed: the Vehicle Routing Problem (VRP) and its variants, and the multi-dimensional Cutting and Packing problems (C&P).

Concerning VRP, successful algorithms were developed by Osman [127], Taillard [143], Xu and Kelly [148], Rego and Roucairol [135], Rochat and Taillard [137], Barbarosoglu and Ozgur [10], Gendreau, Hertz and Laporte [58] (the TABUROUTE algorithm) and Toth and Vigo [145] (the Granular Tabu Search).

For (C&P), TS were proposed by Lodi, Martello and Vigo [112, 114] and Lodi [107] for the two-dimensional Bin Packing Problem, and by Iori, Martello and Monaci [97] for the two-dimensional Strip Packing Problem.

1.2 Scatter Search and Path Relinking

Scatter Search (SS) is a population-based method that has recently shown to produce good results for solving combinatorial and non linear optimization problems. It is based on formulations originally proposed in the 1960s (see, e.g., Glover [62, 63]) for combining decision rules and problem constraints, and uses strategies for combining solution variables related to the specific problem.

Path Relinking (PR) has been suggested as an approach to integrate intensification and diversification strategies in a search scheme that guides the search from a starting solution towards some guiding solutions. The approach may be viewed as a strategy that seeks to incorporate attributes of high quality solutions, by creating inducements to favor these attributes in the moves selected for the creation of new solutions. In this section we examine SS and PR strategies separately, by describing the features that set them apart from other evolutionary approaches (see, e.g., following Section 1.3), and that offer opportunities for creating flexible and effective methods for optimization.

1.2.1 Scatter Search

In a metaheuristic classification, SS may be viewed as an evolutionary population algorithm, that constructs solutions by combining others in some specific way. It derives its foundations from strategies originally proposed in the '60s for combining decision rules and constraints in the context of integer programming. The aim of SS is to enable the implementation of solution procedures that can derive new solutions from combinations of elements of previous solutions, in order to create an effective search framework .

SS is designed to operate on a set of points, called *reference points*, which constitute good solutions obtained from previous solution efforts. Notably, the basis for defining "good" includes special criteria, such as diversity, that go beyond the simple objective function value. The approach generates combinations of the reference points in order to create new points, each of which is then checked for feasibility and possibly used as a starting point for intensification. The combinations are usually generalized forms of linear combinations, accompanied by ideas used to enforce feasibility conditions, including those of discreteness (see, e.g., Glover [64]).

The SS process of combination of the solutions is organized to (i) capture information not contained separately in the original points, (ii) use of auxiliary heuristic intensification methods, in order to evaluate the combinations produced and to actively generates new points, and (iii) make dedicated use of strategies instead of randomization. SS basically consists of five methods, here described:

1. A *Diversification Generation Method*: to generate a collection of diverse (among them) trial solutions, using one or more criteria for the evaluation of such diversity degree.

2. An *Intensification Method*: to transform a solution into one or more improved solutions. Input and output solutions are not required to be feasible, though the output solutions are typically feasible. If the input trial solution is not improved, as a result of the application of this method, the improved solution is considered to be the same as the input one.

3. A *Reference Set Update Method*: to build and maintain a *reference set* consisting of the R "best" solutions found (where the value "R" is typically small, e.g., no more than 20), organized to provide efficient accessing by other parts of the solution procedure. Several alternative criteria may be used to add solutions to and delete solutions from the reference set. Typically a *dynamic* update method consists in including a better solution (in terms of value of the objective function of degree of diversity) in the reference set as soon as this solution is found, while in the *static* update method, the complete set of operations of one iteration is computed and after that the new solutions are checked in order to enter the reference set. Glover, Laguna and Martí [73] suggest the use of the dynamic method, because of computational evidence of better performance than the static one.

4. A *Subset Generation Method*: to operate on the reference set, to produce a subset of its solutions, as a basis for creating combined solutions. The most common subset generation method is to generate all pairs of reference solutions (i.e., all subsets of size 2), but also other similar generations have been used with success in various applications. Another approach, defined *multiple solution method* in the literature (see, e.g., following Chapter II), consists in generating: (i) all 2-element subsets; (ii) the 3-element subsets that are obtained by augmenting each 2-element subset to include the best solution not already belonging to it; (iii) the 4-element subsets that are obtained by augmenting each 3-element subset to include the best solution not already belonging to it; (iv) the i-element subsets (for $i = 5, \ldots \alpha + \beta$) consisting of the best i elements.

5. A *Solution Combination Method*: to transform a given subset of solutions produced by the subset generation method into one or more combined solutions. The combination method is analogous to the crossover operator in genetic algorithms (see following Section 1.3) although it should be capable of combining more than two solutions (it is worth noting that the combination process proposed in the original SS paper included forms of "crossover" not originally envisioned in the GA literature).

The basic procedure starts with the creation of an initial large set of solutions P, using the diversification generation method. Then a certain subset of solutions is extracted from P and is used to create the reference set R. The size of P is typically 10 times the size of R. Initially the reference set consists of α distinct solutions of high quality and β distinct solutions with high diversity. Then, a search is executed by iteratively creating a family F of subsets of R, by applying the combination method to each component of F for creating new solutions, by evaluating the quality and diversity of each new solution, and by eventually inserting the new solution in the reference set. The procedure is outlined as follows:

1. Generate a starting set P of solutions, through the diversification generation method. (Possibly improve each of them through intensification).

2. Associate with each solution a positive integer value that describes its quality and a positive integer value that describes its diversity.

3. Create a reference set $R = R_\alpha + R_\beta$ of distinct solutions by including in R_α the α solutions of P with highest quality, and in R_β the β solutions of P with highest diversity.

4. Evolve the reference set R through the following steps:

   a. Subset generation: generate a family F of subsets of R.

   b. **while** $F \neq \emptyset$ **do**
      Combination: extract a subset from F and apply
      the combination method to obtain a solution s;
      improve s through intensification;
      execute the reference set update on R
      endwhile;

   c. **if** stopping criteria are not met **then go to** a.

The procedure usually terminates when (i) a proof of optimality is given for the best solution (by comparing with a lower bound), (ii) a maximum time limit or number of iterations is reached, or (iii) when a certain number of iterations has passed since the last improvement of the incumbent solution.

This basic design idea can be expanded and improved in different ways. The SS methodology is very flexible, since each of its elements can be implemented in a variety of ways and degrees of sophistication. Different improvements and designs from this basic SS algorithm are given in Glover [69], Glover, Laguna and Martí [73] and Laguna [101].

1.2.2 Path Relinking

In any search method, one of the main goals is to create a balance between search intensification and search diversification. For this purpose, PR has been suggested as an approach to integrate intensification and diversification strategies (see, e.g., Glover and Laguna [71]). Some features that have been added to SS are also captured by the PR framework. This approach generates new solutions by exploring trajectories that connect high-quality solutions, by starting from one of these solutions, called the initial solution, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by carefully selecting moves that introduce attributes contained in the guiding solutions.

The approach may be viewed as a strategy that seeks to incorporate attributes of high quality solutions, by favoring these attributes in the moves selected. However, instead of using an inducement that merely encourages the inclusion of such attributes, the PR approach subordinates other considerations to the goal of choosing moves that introduce the attributes of the guiding solutions, in order to create a good attribute composition in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from a restricted set (i.e., the set of moves currently available, incorporating a maximum number, or a maximum weighted value, of the attributes of the guiding solutions).

The approach is called PR either by virtue of generating a new path between solutions previously linked by a series of moves executed during a search, or by generating a path between solutions previously linked to other solutions but not to each other.

1.2.3 Some successful applications of Scatter Search and Path Relinking

For a survey on successful applications of SS and PR to combinatorial optimization problems, the interested reader is referred to, e.g, Glover [62, 63] and Glover, Laguna and Martí [72].

Some problems that have been addressed by SS and PR need some more words. The *linear ordering problem* consists in finding a permutation of columns and rows that maximizes the weights in the upper triangle of a given matrix. A SS was proposed by Campos, Laguna and Martí [21], obtaining the best quality results with respect to other heuristic and metaheuristic techniques even if in longer computational times.

Given an undirected graph, the *graph $k-$ coloring problem* calls for the partitioning of the vertices in k color classes, for which no couple of vertices linked by an edge belongs to the same class. It has been addressed by Hamiez and Hao [83], using in the intensification phase ideas from a previous Tabu Search algorithm.

Given an undirected graph G, a subgraph G_1 is said to be complete, or equivalently to be a clique, if it exists an edge for each pair of vertices in G_1. The objective of the *maximum clique problem* is to find the maximal clique (i.e., a clique not included in any other clique) of maximum cardinality. For this problem Cavique, Rego and Temido [23] proposed a SS, in which the combinations were selected according to some weights obtained through "filtering vectors", and the intensification phase was obtained through a Tabu Search approach, obtaining encouraging results.

Concerning the scheduling theories, the problems of $P||C_{max}$ and K_i- partitioning call for the minimization of the maximum completion time of the working operations on a set of jobs, performed by a set of parallel processors, and are a generalization of the well known $P||C_{max}$ problem. They have been addressed with SS by Dell'Amico, Iori and Martello [39], and by Dell'Amico et al. [40], who proposed two similar approaches, in which intensification was obtained through two-opt and other tailored strategies, the resulting approaches are the state of the art for the heuristic solution of these problems.

1.3 Genetic Algorithms

The *Genetic Algorithms* (GAs) where first introduced by Holland [91] in 1975 and became one of the most largely used technique for efficiently solving combinatorial optimization, linear and non-linear problems. Nowadays also the terms *evolutionary computing* and *evolutionary algorithms* are commonly used to refer to this field of study. Other terms, namely *evolution strategies* and *evolutionary programming*, were introduced already in the 1960s (previously to Holland's work).

The common idea behind these terms is the use of *mutation* and *selection*, the concepts of the Darwinian evolution theory, in order to perform a search process in the solution space. Although the 1975 was an important year for the publishing of the seminal works on GAs, the evolutionary computing took off only in the 1980s, especially with the work of other Ph.D. students of Holland, as, e.g., Goldberg [75], who successfully applied GAs to different optimization fields.

1.3.1 Basic Features

One of the distinctive features of the GAs is to allow the separation of the *representation* of the problem from the actual variables in which it was originally formulated, distinguishing

between the *genotype* (i.e., the encoded representation of the variables) and the *phenotype* (the set of variables themselves).

In practice, when we are given an objective function $f(x)$ to be minimized, where x is an $n-$ dimensional vector of variables, we create a mapping between x and a ceratin string s of length ℓ, made up by 0-1 encoding or integer encoding (or generally by a given alphabet). The length ℓ of the string depends from the decision variable x and from the alphabet. Its elements are defined *genes*, and the values that these elements can take *alleles*. In general, this is defined as the phenotype - genotype mapping, and it is one of the central point in the development of a good GA (usually this mapping is constituted by a bijection). A good description of the ideas that can be used to develop this mapping are presented by, e.g., Reeves [132].

The original motivation for GAs was the biological analogy: if the natural process of selecting the best individuals for reproduction and for the creation of new individuals managed to develop strong species adapted to their environments, would it manage to find good solutions also for optimization problems? In the selective breeding of plants and animals, offspring are sought to receive certain desiderable characteristics, determined by the genetic combination of the parents' chromosomes. In the case of GAs, a population of strings (usually referred to, in the literature of evolutionary programming, as *chromosomes)* is used in order to obtain genetic recombination through mutation and *crossover operators*. Based on the evaluation of a certain criteria of goodness of the solution, not only dependent from the value of the objective function, and defined as *fitness*, the strings have a lower or higher probability of being selected for reproduction. Fitness is a central key to GAs and it is usually defined in order to avoid too flat search space, but to create valleys and mountains, so as to guide properly the search process.

When two or more parents are selected for reproduction, the operators of crossover and mutation are used. Crossover is a matter of replacing some of the genes in one parent, with some other genes of the other parent, consequently producing an offspring. Good crossover operators where proposed by, e.g., Reeves [134],Davis [35, 36], Goldberg and Lingle [75], Jakobs [99] and Poon and Carter [129]. Mutation is instead applied to a single chromosome, where some of the genes are randomly selected and the corresponding allele values are changed.

1.3.2 Additional Features

In this section a description (quite superficial for sake of conciseness) is given for some parameters among those that must be correctly defined in order to achieve a good GA approach.

After the definition of the encoding of an individual, the first relevant point to consider is the size and the composition of the *initial population*. Concerning the size, it is important to find a good compromise between efficiency (in the terms of computational complexity and time) and efficacy (in terms of quality of the solutions achieved) of the GA. As to how the population is chosen, it is commonly assumed a random creation, but there are several approaches that use heuristic techniques in order to produce a first population containing already good quality solutions. The size of the population can remain unchanged in the following generations (*steady state*) or vary according to different criteria.

Concerning the *reproduction* of the individuals, one can use crossover and mutation at the same time, or use only one of them, or taking into account other tailored mechanism dependent from the problem addressed. Usually the crossover is always applied, and the mutation has just a low probability of being selected, since empirical studies prove that, with

higher probabilities, mutation has the negative effect of reducing the average solution value of the population and disallowing the achievement of new good solutions.

It is worth noting that the *diversification* phase, common to almost all the metaheuristic approaches, can be easily obtained in GAs with a correct use of the *mutation*. An alternative approach is the use an *immigration* theory, that operates by including in the new generations individuals either randomly created, or coming from areas not frequently searched during the execution of the algorithm (the *history* of the evolution has thus to be memorized).

The *intensification* is relevant for any successful evolutionary approach. It can be obtained in GAs through normal procedures of local search or through what is known in evolutionary computing as *hybridization*. It consists in using a heuristic which can be called at each time a new individual is created. Hybridization proved to be very useful, if not necessary, for efficiently addressing many optimization problems.

The basic idea for the *selection* of the parents to be mated is that it should be related to fitness, and the original scheme for its implementation is commonly known as the *roulette-wheel* method. It uses a probability distribution for selection, in which the selection probability of a given string is proportional to its fitness. Another method, known as the *stochastic universal selection*, proved to be particularly effective for reducing the high stochastic variability of the roulette-wheel method (for details on this approach see the original work by the author: Baker [9]). Another approach is the *tournament selection*, in which a subset of parents is randomly chosen and the best among them is used for parent selection.

1.3.3 Some successful applications of Genetic Algorithms

There is an enormous number of papers that present GAs for combinatorial optimization problems (see, e.g., Reeves [134, 133] and Davis [36]). For multi-dimensional packing problems, successful approaches were proposed for the two-dimensional knapsack problem by Beasley [14], with a non-linear formulation of the problem, and by Hadjiconstantinou and Iori [81] with a hybridized approach.

For the Traveling Salesman Problem, a first attempt with a GA by Goldberg and Lingle [75], was later outperformed by Grefenstette [79], who included in his GA a specific knowledge of the problem and a mutation operator that rapidly performed a neighborhood structure. Another later approach by Suh and Van Gucht [141] showed better results, by hybridizing the GA with heuristics, two-opt and simulated annealing.

1.4 Simulated Annealing

Simulated Annealing (SA) became in these years a very popular metaheuristic for its convergence properties and its use of hill-climbing moves to escape from local optima. Many surveys are available on this technique, among which we cite Henderson, Jacobson and Johnson [86], Aarts and Lenstra [2] and Aarts, Korst and Van Laarhoven [1].

SA is so named because of its analogy to the process of physical annealing with solids, in which a crystalline solid is heated and then cooled slowly until it reaches its most regular crystal lattice configuration (i.e., its minimum state energy). When the cooling schedule is sufficiently slow, the resulting structure is free of crystal defects.

SA establishes a connection between this thermodynamic process and the search for heuristic solutions for optimization problems. The algorithm starts from a heuristic solution and at each iteration tries to improve its value. Improving solutions are always accepted, while

non-improving solutions are accepted only under given conditions. The probability of accepting non-improving moves is indeed proportional to a parameter, defined *temperature* in SA literature, which topically decreases during the execution of the approach. The key feature is that SA provides a means to escape from poor local optima, by allowing hill-climbing moves. As the temperature decreases, tending toward zero, the worsening moves are accepted with less frequency, and the solution tends to a (local or possibly global) optimum.

1.4.1 A simple Template for Simulated Annealing

Define S as the search space of the current problem, x the vector of decision variables, and $f(x)$ the objective function. Define also a heuristic solution at a given iteration as s, and s' the candidate new solution, which can be selected in the neighborhood $N(s)$ of s. The candidate s' is selected according to a probability $P_{s'}$:

$$P_{s'} = \begin{cases} exp(-(f(s') - f(s))/t_k) & \text{if (f(s')-f(s) } > 0); \\ 1 & \text{otherwise} \end{cases} \tag{1.1}$$

This acceptance probability is the basic element of the search mechanism in SA. If the temperature t_k, where k is the number of the iteration, is reduced sufficiently slowly, then the system can reach an equilibrium.

As common for the other metaheuristic procedures, also for SA an intensification phase can improve consistently the quality of the solution obtained. The diversification is indeed included in the mechanism itself of the search. When, especially during the first iterations, the temperature is high, the probability of accepting solutions that, in spite of a worst objective value, leads to different areas of the search space is quite high. When such approach showed to be not sufficient for moving from poor areas, SA was also implemented with a multi start approach.

The convergence of such approach is proved for a computing time tending to infinite and it is based on homogeneous or inhomogeneous Markov chain. It was noted, however, that the idea of simple choosing randomly new solutions can as well lead to the convergence to the optimum in an infinite time, and thus the proof of convergence for SA is not particularly relevant.

1.5 Ant Colony Optimization

The Ant Colony Optimization metaheuristic (ACO) is a recent technique used for solving combinatorial optimization problems. The source from which ACO takes inspiration is the use by ants of pheromones as a communication method (laying it on the ground as a guide for the next ants). In analogy to the biological example, ACO is based on the indirect communication of a colony of simple agents, called artificial ants, mediated by artificial pheromone trials.

The pheromone trial in ACO is a distributed numerical information, which is used by the ants for probabilistically constructing solutions to the problem being solved and updated, by the same ants, during the execution of the process. In spite of many cases in which ACO could not reach the results obtained by other metaheuristic techniques, the approach is still being used to address several optimization problems, among which quadratic assignment, vehicle routing, sequential ordering and scheduling. For recent surveys and presentations of

frameworks for ACO see Dorigo and Stützle [45], Dorigo and Di Caro [43] and Dorigo, Di Caro and Gambardella [44].

1.5.1 Features of the Ant Colony Optimization Algorithm

Artificial ants used in ACO are solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions by taking into account both heuristic information on the problem instance being solved, and pheromone trails which change dynamically at run-time to reflect the agents' acquired search experience.

A stochastic component in ACO allows the ants to build a variety of different solutions. At the same time, the use of heuristic information can guide the ants towards the most promising areas of the search space. Moreover, the ants' experience accumulated in the search process can be used to influence the solution construction in future iterations of the algorithm. Additionally, the use of a colony of ants can give the algorithm increased robustness and in many ACO applications the collective interaction of a population of agents is needed to efficiently solve a problem.

Given a representation of the problem as a construction graph, in which the vertices are the *components* (characteristics depending from the formulation used to represent the problem) and the edges are the *connections* between two components, the ants move from one vertex to the other, leading to the construction of the solutions.

Ants do not move arbitrarily on the graph, but rather follow a construction policy which is function of the problem constraints. If necessary or desiderable, they can construct infeasible solutions. Components and connections have an associated pheromone trail, which encodes a long term memory concerning the whole process and is iteratively updated. It is important to note that ants move independently one from the other. A single ant has a low probability of finding the global optimum, but the collection of the, generally high, number of ants composing the colony has an overall much stronger probability. The approach obtained by the movements of the ants through adjacent states of the graph, allow the ants to construct solutions. The evaluation of this (eventually) partial solutions is used in order to update the pheromone.

Two additional features can be included in the approach. The use of *pheromone evaporation* can be useful in order to reduce the possibility of a fast convergence to a poor local optimum, and consists in reducing the pheromone left on the single components or connections during the time. In a certain sense it represents the diversification phase common to other metaheuristic techniques. The *intensification* is obtained through processes that cannot be operated by single ants, typically collecting information at a central level, and passing these information to a tailored intensification procedure.

1.6 Variable Neighborhood Search

The Variable Neighborhood Search (VNS) is a metaheuristic proposed recently (see, e.g., Mladenović, Labbé and Hansen [124]) and is based upon a simple principle: systematically changes in the size and type of neighborhood during the search process.

Defining as N_k, for $Kk = 1, \ldots, k_{max}$, the set of selected neighborhoods for the problem addressed, the idea of moving from one neighborhood to another is used in order to escape from poor local optima (enlarging the size of the neighborhood), or intensifying the search in promising areas (reducing the size of the neighborhood). Concerning this aspect, it is worth

noting that a local optimum with respect to a neighborhood is not necessary a local optimum with respect to another neighborhood.

However, the time (or iteration) in which a change of neighborhood has to be made is not clear. The changes between one neighborhood and the other can be performed either deterministically, either stochastically or both. If the deterministic way is chosen, the resulting algorithm is defined as the *Variable Neighborhood Descent* and has been successfully applied to several problems.

The *Reduced VNS* is instead a version of VNS in which the neighborhoods are randomly selected from the N_k without applying local search, and is used for large scale problems, for which intensification of the search would be too time consuming.

Finally, the *Basic VNS* is constituted by a mix of the two previous, in which the choice of the neighborhood is either deterministic or stochastic. It can be outlined a s follows:

1 Select the set of neighborhood structures N_k

2 Generate an initial solution x

3 Repeat the following steps until $k = k_{max}$

- 3.1 Shaking: generate a point x' at random from the kth neighborhood of x
- 3.2 Local search: apply some local search method to x' in order to obtain a local optimum x''
- 3.3 Move or not: if x'' is better than x then move to the new solution and set $k = 1$, otherwise keep x and set $k = k + 1$

Note that cycling is avoided by using randomization. The stopping condition are the usual ones for metaheuristic techniques (i.e., maximum elapsed time, maximum number of iterations or proof of optimality of the incumbent solution). For a detailed introduction to VNS, see, e.g., Hansen and Mladenović [84].

1.7 Other

For sake of conciseness we revise in a unique section other meta-heuristic techniques that proved to be useful for solving combinatorial optimization problems. We refer to Guided Local Search, Multi-Start methods, Greedy Randomized Adaptive Search Procedure, Neural Networks and Memetic Algorithms.

1.7.1 Guided Local Search

Guided Local Search (GLS) is a metaheuristic that tries to avoid the problem of remaining stuck in some local optima, by assigning penalty factors to given characteristics of the solutions (for a general introduction to the technique, see, e.g., Voudouris and Tsang [146]).

To apply GLS to the problem addressed, one has to define a set of features for the candidate solutions. GLS then starts by finding a first heuristic solution and applying Local Search (LS) to this solution. When LS remains trapped in a local minimum, certain features are selected and penalized. The new resulting objective function is then minimized by applying again LS, and the process is iterated until some stopping criterion are met. The novelty of GLS lies in

the way in which it selects features and how it penalizes them, by allowing the search effort to be distributed in the search space.

The penalty factors are initially set to 0 and are updated only when the LS finds a local minimum. Given an objective function $f(s)$ that maps every solution s to a numerical value, GLS minimizes a modified objective function $g(s)$ such that:

$$g(s) = f(s) + \alpha \cdot \sum_{i \in I} p_i x_i(s) \tag{1.2}$$

where α is a parameter, I is the set of features, p_i the value of the penalty and $x_i(s)$ a 0-1 variable, taking value 1 iff solution s contains feature i.

The features are penalized according to a *utility* function that varies according to the different approaches, but that remains directly proportional to the cost induced by the feature in the solution, and inversely proportional to the frequency of penalization of the feature.

A good quality that usually appears in GLS is that the approach is not particularly sensitive with respect to the parameter α, and that for many problems the description of the features directly come with the objective function.

1.7.2 Multi-Start Methods

The search processes implemented by the metaheuristic techniques usually require diversification strategies in order to escape from poor local optima. Without this diversification, these methods can become too localized in a small area of the solution space, and eventually leading to a poor solution. A usual way of obtaining this diversification is to consider the history of the search process, or the frequency in which a solution was reached or similar approaches. An alternative way is to stop the algorithm and to restart it from a different point of the space.

Multi-Start methods (MS) have been created in order to implement this latter technique. They are used in order to guide the construction of new solutions in a long term horizont and they provide an appropriate mechanism for diversification (see, e.g., Martí [121]).

MS are generally composed by two phases: a first one in which the solution is generated, and a second one in which it is improved through a general local search or metaheuristic technique. Many techniques, coming from general frameworks or directly tailored to the problems are available for the first step. Concerning the second step it is of interest to note that some successful approaches tend to anticipate the use of local search and to adapt it directly to the partially constructed solution. One of the most well known MS is the Greedy Randomized Adaptive Search Procedure (see following Section 1.7.3).

A possible classification of the different MS approaches lies on the definitions of its main features, mainly: use of *memory*, *randomization* against *diversity measure* and *degree of rebuild*, that indicates the quantity of elements that remain unchanged from one iteration to the other.

1.7.3 Greedy Randomized Adaptive Search Procedure

The Greedy Randomized Adaptive Search Procedure (GRASP for short) is a multi start metaheuristic optimization technique that consists basically of two steps: (i) constructing a feasible solution and (ii) improving the solution through local search. These two steps are iteratively performed, starting from different points (i.e., solutions to step (i)) and the best

solution is kept in memory and finally returned in output. Many mechanisms and tricks have been implemented in order to improve the efficiency of this simple but well performing approach, among which we cite *Reactive GRASP* (see, e.g., Resende and Ribeiro [136]), cost perturbations, memory and history of the search, learning and local search on partially constructed solutions.

The first step of GRASP consists of a *greedy randomized construction* of a solution, and is executed by considering a *candidate list* of elements that can enter the partial solution. The process starts with an empty solution, considers the complete candidate list and selects an element taking into account the cost of insertion of this element in the solution, a randomization factor and eventually other criteria (as, e.g., history of the search and frequency of selection of the element). The element is inserted in the partial solution and the candidate list is reduced by considering only those elements that could be inserted in the partial solution without leading to an infeasibility. The process is iterated until a complete, hopefully feasible, solution is created. The solution is then given to Step (ii) of the algorithm, that computes a local search, reaching a local optimum of the search space. The local search procedure can be a standard one, with a tailored definition of the neighborhood structure and of the search.

An extensive survey on GRASP literature and on its successive applications to combinatorial optimization problems is given in Festa and Resende [54].

1.7.4 Neural Networks

Artificial Neural Networks (ANNs) were originally proposed for providing a new approach for information processing, in alternative to the standard algorithmic computing. They date back to 1943 (see Mc Culloch and Pitts [122]) and since then have been applied to a great variety of problems.

ANNs are capable of internal developing information processing capabilities, by considering information coming from the problem addressed. After the seminal work in 1943, where ANNs were shown to be able to learn mathematical and logical functions, important successes were obtained during the '60s, with the development of the first neurocomputers. After a period of relatively discouragement, these techniques were again reused for optimization from the '80s.

Hopfield [93] described a way of modeling a system of neurons capable of performing computational tasks, using a collection of binary neurons and a stochastic updating algorithm. This model was used to address the Traveling Salesman Problem, and was able to solve instances with a limited number of clients. For a recent survey, see, e.g., Potvin and Smith [130]).

1.7.5 Memetic Algorithms

Memetic Algorithms (MAs) represent a broad class of metaheuristics. The main idea of MA approaches is to combine the effective search method of Genetic Algorithms (GAs, see Section 1.3), with the use of specific information related to the optimization problem addressed.

As for the evolutionary approaches, MAs use a population of solutions that are combined together through crossover and mutation in order to produce new solutions. The intensification phase is obtained by incorporating heuristics, approximation algorithms, local search, truncated exact methods and other techniques tailored to the solution of the specific problem. The term MA is particularly used when the intensification phase is performed with the use of

another nested metaheuristic. Typically MAs present a GA in which a nested Tabu Search or Simulated Annealing is used.

A list of successful approaches of MAs to combinatorial optimization problems is given by Moscato and Cotta [126]. Regarding the multi-dimensional packing problems, MAs have been developed by Beasley [14], where the intensification was obtained through heuristics, and Iori, Martello and Monaci [97], who adopted a nested Tabu Search in their GA. For the Traveling Salesman Problem, MAs were presented by Grefenstette [79] and Suh and Van Gucht [141], who used a nested Simulated Annealing.

Part II

Algorithms for Scheduling Problems

Chapter 2

On the Cardinality Constrained $P||C_{\max}$ Problem

[1]

We consider the generalization of the classical $P||C_{\max}$ problem (assign n jobs to m identical parallel processors by minimizing the makespan) arising when the number of jobs that can be assigned to each processor cannot exceed a given integer k. The problem is strongly NP-hard for any fixed $k > 2$. We briefly survey lower and upper bounds from the literature. We introduce greedy heuristics, local search and a scatter search approach. The effectiveness of these approaches is evaluated through extensive computational comparison with a depth-first branch-and-bound algorithm that includes new lower bounds and dominance criteria.

Key words: Scheduling, parallel processor, cardinality constraint, scatter search.

2.1 Introduction

Given n jobs, each characterized by a processing time p_j $(j = 1, \ldots, n)$, and m identical parallel processors, each of which can process at most one job at a time, consider the problem of assigning each job to a processor so that the maximum completion time of a job (*makespan*) is minimized. The problem is denoted as $P||C_{\max}$ in the three field notation by Graham et al. [78] and is known to be strongly NP-hard. The problem can also be seen as the 'dual' of another famous combinatorial optimization problem that will be also considered in the following: The *Bin Packing Problem*, calling for the partitioning of a given set of n items, each having an associated weight p_j, into the minimum number of subsets (*bins*) such that the total weight in each subset does not exceed a given *capacity* c. It is then clear that, by determining the minimum c value such that a bin packing instance has an m-subset solution, we also solve the associated $P||C_{\max}$ instance.

In this paper we consider a generalization of $P||C_{\max}$ in which an additional constraint imposes that the number of jobs that can be assigned to a processor is at most k, denoted as $P|\# \leq k|C_{\max}$. The problem is strongly NP-hard for any fixed $k > 2$ (see Dell'Amico and Martello [38]), while for $k = 2$ it is solvable in $O(n \log n)$ time by sorting the jobs according to non increasing processing time and assigning job j to processor j for $j = 1, \ldots, m$, and job

[1] The results of this chapter appears in: M. Dell'Amico, M. Iori and S. Martello, Heuristic Algorithms and Scatter Search for the Cardinality Constrained $P||C_{\max}$ Problem, *Journal of Heuristics*, 10: 169-204, 2004 [39].

$m+j$ to processor $m-j+1$ for $j=1,\ldots,n-m$. We assume that the processing times p_j are non-negative integers, that $2 \leq m$, $2m \leq n$ and that $n \leq mk$.

Possible applications of $P|\#\leq k|C_{\max}$ arise when m processors (e.g., cells of a Flexible Manufacturing System, robots of an assembly line), have to perform n different types of operation. In real world contexts, each processor can have a limit k on the number of different types of operation he can be perform, coming, e.g., from the capacity of the cell tool magazine or the number of robot feeders.

Lower bounds for $P|\#\leq k|C_{\max}$ were presented by Dell'Amico and Martello [38]. The special case arising when $n=mk$, usually denoted as *k-partitioning problem* (*KPP*), was studied by Babel, Kellerer and Kotov [5]. Note that an instance of $P|\#\leq k|C_{\max}$ can be transformed into an instance of k-partitioning by adding $n-mk$ dummy jobs with zero processing time.

In Section 2.2 we review lower bounds from the literature. In Section 2.3 we present greedy heuristics and in Section 2.4 a scatter search algorithm with local search procedures. In Section 2.5 we introduce an enumerative algorithm, together with lower bounds and dominance criteria. Finally, in Section 2.6, the effectiveness of our approaches is tested through extensive computational experiments performed both on random data sets and real world instances.

2.2 Lower bounds from the literature

Problem $P|\#\leq k|C_{\max}$ can be formally stated as:

$$\min z \tag{2.1}$$

$$\sum_{j=1}^{n} p_j x_{ij} \leq z \quad (i=1,\ldots,m) \tag{2.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (j=1,\ldots,n) \tag{2.3}$$

$$\sum_{j=1}^{n} x_{ij} \leq k \quad (i=1,\ldots,m) \tag{2.4}$$

$$x_{ij} \in \{0,1\} \quad (i=1,\ldots,m; j=1,\ldots,n) \tag{2.5}$$

where z is the optimum makespan, and x_{ij} takes the value 1 iff job j is assigned to processor i. Without loss of generality we assume that the jobs are sorted by non-increasing value of their processing time. Since any lower bound for $P||C_{\max}$ is obviously valid for $P|\#\leq k|C_{\max}$, we will both consider bounds adapted from $P||C_{\max}$ and *KPP*, and bounds that explicitly take into account the new constraint.

Dell'Amico and Martello [41, 38] proposed a simple lower bound,

$$L_2 = \max\left(\left\lceil \frac{1}{m}\sum_{j=1}^{n} p_j \right\rceil, \max_j\{p_j\}, p_m + p_{m+1}\right) \tag{2.6}$$

given by the maximum among the solution value of the continuous relaxation, the largest processing time of a job and the minimum makespan of a processor when no less than $m+1$ jobs have to be scheduled. When $n > m(k-1)$, the bound was strengthened by observing

that at least one machine must process k jobs among the first (largest) $m(k-1)+1$ ones: By considering the k smallest such jobs we obtain:

$$\widetilde{L}_2 = \max\left(L_2, \sum_{j=(m-1)(k-1)+1}^{m(k-1)+1} p_j\right) \tag{2.7}$$

We note that, in the special case of *KPP* (where $n = mk$), the latter bound can be further improved by also considering a lower bound on the makespan of the processor that handles the largest job:

$$\widetilde{L}'_2 = \max\left(\widetilde{L}_2, p_1 + \sum_{j=n-k+2}^{n} p_j\right) \tag{2.8}$$

All the above bounds can be computed in $O(n)$ time. The scatter search heuristic of Section 2.4 and the enumerative algorithm of Section 2.5 make also use of other more complex bounds from the literature, for which we just give an intuitive explanation, referring the reader to the specific papers. In particular:

- L_3: This bound was developed by Dell'Amico and Martello [41] for $P||C_{\max}$, and is based on a partition of the jobs, according to their processing time, determined by a threshold value $\overline{p}$. Each $\overline{p}$ value produces a valid lower bound, and L_3, the maximum among them, is determined in a time that is a pseudo-polynomial function of an upper bound on the optimum makespan.

- L_3^k: Developed for $P|\#\leq k|C_{\max}$ by Dell'Amico and Martello [38], this bound too is based on thresholds and job partitioning, and has pseudo-polynomial time complexity.

- L_{BKK}: polynomial time bound proposed by Babel, Kellerer and Kotov [5] for the k-partitioning problem, given by the maximum among three bounds obtained from continuous relaxations and considerations related to the famous LPT heuristic for $P||C_{\max}$ (see below, Section 2.3).

- L_{HS}: Consider the associated bin packing instance described in the Introduction. Hochbaum and Schmoys [90] have proposed an approximation algorithm that, for a given capacity c, solves, in linear time, a relaxed problem that provides a lower bound $m(c)$ on the number of bins needed in any feasible solution. We then obtain $L_{HS} = \max\{c+1 : m(c) > m\}$, that is computed in psuedo-polinomial time through binary search on c.

2.3 Heuristic algorithms

In this section we first describe heuristic algorithms for $P||C_{\max}$, and then heuristics obtained by modifying them so as to handle the cardinality constraint. In the following we denote by $C(i)$ the current completion time of processor i, by $k(i)$ the number of jobs currently assigned to i, by L the best lower bound value obtained and by z the incumbent solution value.

2.3.1 Heuristic algorithms for $P||C_{\max}$

Many approximation algorithms are available for $P||C_{\max}$ (see, e.g., the surveys by Lawler et al. [104], Hoogeveen, Lenstra and van de Velde [92], Mokotoff [125]).

A very popular approach is the *List Scheduling* (*LS*) approximation algorithm (see Graham [77]), that sequentially assigns the jobs, in some pre-specified order, to the processor i with minimum $C(i)$, without introducing idle times. If we apply *LS* to a job list sorted by non-increasing p_j value, we obtain the so called *Longest Processing Time* (*LPT*) algorithm, which often produces good approximate solutions, (see also its probabilistic analysis in Coffman, Lueker and Rinnooy Kan [29]).

A different approach is the *Multi Fit* (*MF*) heuristic (see Coffman, Garey and Johnson [28]) that finds the smallest value u for which an approximate solution to an associate bin packing problem instance uses no more than m bins of capacity u.

Another effective $P||C_{\max}$ heuristic is the *Multi Subset* (*MS*) algorithm by Dell'Amico and Martello [41]. Given n items j with weights p_j ($j = 1, \ldots, n$), and a prefixed capacity c, the *Subset-Sum Problem* (*SSP*) is to find a subset of the items whose total weight is closest to, without exceeding, c. Given a lower bound L on the $P||C_{\max}$ solution value, algorithm *MS* works as follows. At iteration i ($i = 1, \ldots, m$), *MS* solves an SSP on the instance induced by the currently unassigned jobs with capacity L, and assigns the resulting job subset to processor i. When all the processors have been considered, the residual unassigned jobs, if any, are assigned through the *LPT* heuristic. The SSP instance considered at each iteration can be solved either exactly (in non-polynomial worst-case time, being the problem NP-hard) or heuristically, through the algorithms in Martello and Toth [117, 118].

2.3.2 Heuristic algorithms for $P|\# \leq k|C_{\max}$

We describe here three heuristics for $P|\# \leq k|C_{\max}$, namely algorithms LPT_k, MS_k and $MS2_k$, obtained by adapting algorithms for $P||C_{\max}$ so as to handle the cardinality constraint.

Algorithm LPT_k was already introduced in [5]: At iteration j ($j = 1, \ldots, n$), job j (the largest unassigned job) is assigned to the processor i with minimum $C(i)$ value among those satisfying $k(i) < k$. Ties are broken by selecting the largest $k(i)$ value.

We derived algorithm MS_k from algorithm *MS* of Section 2.3.1. In the iterative phase, the associated SSP instance is solved by only considering subsets of cardinality not greater than k. In the second phase, the residual unassigned jobs are assigned through LPT_k. The specialized algorithm for SSP was obtained by adapting algorithm G^2 by Martello and Toth [117]. Algorithm G^2 is an $O(n^2)$ time heuristic for SSP that selects the best solution among $O(n)$ solutions produced by a greedy algorithm executed on items sets $\{1, \ldots, n\}$, $\{2, \ldots, n\}$, ..., respectively. The greedy algorithm for SSP iteratively considers all the items: The next item is inserted into the current subset if the capacity is not exceeded. In order to adapt it, it is then enough to terminate its execution as soon as the cardinality limit has been reached.

Algorithm $MS2_k$ is based on partial enumeration and algorithm MS_k above. We start by generating the first ℓ levels of our branch-and-bound algorithm (see below, Section 2.5): The leaves of the resulting branch-decision tree represent all non-dominated solutions involving the ℓ largest jobs. The current lower bound value L is then possibly improved by the smallest lower bound associated with a leaf. For each leaf, we complete the associated partial solution through an adaptation of MS_k that:

(i) only uses items $\{\ell + 1, \ldots, n\}$;

(ii) at each iteration of the first phase (i.e., at each solution of an induced SSP solution), decreases the available capacity and the maximum cardinality of the current processor i by the total weight $C(i)$ and number of jobs $k(i)$, respectively, currently assigned to i in the leaf solution;

(iii) assigns the residual unassigned jobs through LPT_k.

The best complete solution obtained from a leaf is finally selected. In our implementation, the value $\ell = 5$ was adopted, based on the outcome of computational experiments.

2.4 Scatter search

This metaheuristic technique derives from strategies proposed in the Sixties for combining decision rules and constraints (see Glover [62, 63]), and was successfully applied to a large set of problems (see, e.g., Glover [68, 69]). The basic idea (see Laguna [101], Glover, Laguna and Martí [73]) is to create a set of solutions (the *reference set*), that guarantees a certain level of "quality" and of "diversity". The iterative process consists in selecting a subset of the reference set, in combining the corresponding solutions, through a tailored strategy, in order to create new solutions, and in improving them through local optimization algorithms. The process is repeated, with the use of diversification techniques, until certain stopping criteria are met.

2.4.1 Local optimization algorithms

In this section we introduce the local search algorithms used within our scatter search approach. All the algorithms receive in input a feasible solution, with processors sorted by non-increasing $C(i)$ value.

Procedure *MOVE*: For each processor i, in order, let j be the largest job currently assigned to i, and execute the following steps:

a. find the first processor $h > i$, if any, such that $k(h) < k$ and $C(h) + p_j < C(i)$, and move job j to h;

b. if no such h exists, let j be the next largest job of i, if any, and go to a.

As soon as a move is executed, the procedure is re-started, until no further move is possible.

Procedure *EXCHANGE*: For each processor i, in order, let j be the largest job currently assigned to i, and execute the following steps:

a. find the first processor $h > i$, if any, such that there is a job q, currently assigned to h, satisfying $p_q < p_j$ and $C(h) - p_q + p_j < C(i)$, and interchange j and q;

b. if no such h exists, let j be the next largest job of i, if any, and go to a.

As soon as an exchange is executed, the procedure is re-started, until no further exchange is possible.

Procedure *REOPT*: For each processor i satisfying $L \leq C(i) < z$, in order, execute the following steps:

a. remove from the instance the jobs currently assigned to i;

b. solve the reduced instance, with $m-1$ processors, through LPT_k followed by *MOVE* and *EXCHANGE*;

c. complete the solution by re-assigning to i the removed jobs.

In addition, the following two improvement procedures are used for *KPP* instances.

Procedure MIX_k: This algorithm adopts a sort of dual strategy with respect to MS_k (see Section 2.3.2). It receives in input a feasible solution and two parameters, $\overline{n}$ and $\overline{k}$ ($1 < \overline{n} < n$, $1 < \overline{k} \leq k$), and creates a new solution as follows:

1. assign the first $\overline{n}$ jobs as in the input solution;

2. sort the processors according to non-increasing $C(i)$ value;
   for $i := 1$ **to** m **do**

   $k' := k - k(i)$;

   if $k' > \overline{k}$ **then**

   assign to i the smallest $k' - \overline{k}$ unassigned jobs;

   $k' := \overline{k}$;

   end if;

   find, through complete enumeration, a set S of k' unassigned jobs, such that $\sum_{s \in S} p_s + C(i)$ is:

   (a) closest to, without exceeding, L, if such an S exists;

   (b) closest to L otherwise;

   end for

Based on our computational experiments, we adopted the values $\overline{k} = 4$ and $\overline{n} = \max\{m, n - 2m\}$ (but $\overline{n} = \max\{m, n - 4m\}$ at the first scatter search iteration).

Procedure $MIX2_k$: This is a variant of MIX_k in which step 1 is replaced by:

1. assign the first $\widetilde{k}$ jobs of each processor as in the input solution;

where $\widetilde{k}$ is a given parameter for which we adopted the value $\widetilde{k} = \max\{0, (k-2)\}$ (but $\widetilde{k} = \max\{0, (k-4)\}$ at the first scatter search iteration).

It is not difficult to adapt both MIX_k and $MIX2_k$ to non-*KPP* instances, although our computational experiments only showed good results for the *KPP* case.

2.4.2 Scatter search strategy

We first outline the main elements of our scatter search approach and then give the details of the various steps.

1. Randomly generate a starting set P of solutions. Improve each of them through *intensification*.
2. Associate with each solution a positive integer value (*fitness*) that describes its "quality".
3. Create a reference set $R = R_\alpha + R_\beta$ of distinct solutions by including in R_α the α solutions of P with highest fitness, and in R_β the β solutions of P with highest *diversity*.
4. Evolve the reference set R through the following steps:
   a. *Subset generation*: generate a family F of subsets of R.
   b. **while** $F \neq \emptyset$ **do**
      Combination: extract a subset from F and apply
      the combination method to obtain a solution s;
      improve s through *intensification*;
      execute the *reference set update* on R
      endwhile;
   c. **if** *stopping criteria* are not met **then go to** a.

In our implementation, the initial set P has size 80, while the reference set R has size 15, with $\alpha = 8$ and $\beta = 7$. The other main features of the approach are:

a. *Intensification.* It consists in executing, in sequence: MIX_k and $MIX2_k$ (only for *KPP* instances), *REOPT*, *MOVE* and *EXCHANGE*.

b. *Fitness.* In order to highlight the differences between solutions that have very close values, we use a fitness function, instead of the value of the solution. This allows us to obtain a less flat search space, and directs the search towards more promising areas. If $z(s)$ is the value of solution s, the correspondent fitness is defined as

$$f(s) = z(s)/(z(s) - L) \tag{2.9}$$

where L denotes the best lower bound value obtained so far.

c. *Diversity.* The diversity of a solution from those in the current reference set is evaluated by considering the $2m$ jobs with larger processing time. For a solution s, let y_{sj} ($j = 1, \ldots, 2m$) be the processor job j is allocated to. The diversity of s is then

$$d(s) = \min_{r \in R} |\{j \in \{1, \ldots, 2m\} : y_{sj} \neq y_{rj}\}| \tag{2.10}$$

d. *Subset generation.* We adopted the multiple solution method (see, e.g., Glover, Laguna and Martí [73]), that generates:
   i. all 2-element subsets;
   ii. the 3-element subsets that are obtained by augmenting each 2-element subset to include the best solution not already belonging to it;

iii. the 4-element subsets that are obtained by augmenting each 3-element subset to include the best solution not already belonging to it;

iv. the i-element subsets (for $i = 5, \ldots \alpha + \beta$) consisting of the best i elements.

e. *Combination.* For a given subset S, we define an $m \times n$ fitness matrix F with $F_{ij} = \sum_{s \in S(i,j)} f(s)$, where $S(i,j) \subseteq S$ is the set of solutions where job j is assigned to processor i. We then select the best among three solutions, each created through a random process that, for $j^* = 1, \ldots, n$, assigns job j^* to processor i^* with probability $F(i^*, j^*) / \sum_{i=1}^{m} F(i, j^*)$: if processor i^* now has k jobs assigned, we set $F(i^*, j) = 0$ for $j = 1, \ldots, n$ (so i^* is not selected at the next iterations). If for the current job j^* we have $F(i, j^*) = 0$ for all i, the job is assigned to the processor with minimum completion time $C(i)$ among those with less than k jobs assigned.

f. *Reference set update.* In order to evolve the reference set R by maintaining a good level of quality and diversity, we adopted the *dynamic reference set update* (see, e.g., Glover, Laguna and Martí [73]). A new solution immediately enters R if its quality is better than that of the worst solution of R_α, or if its diversity is greater than that of less different solution of R_β. Solutions that are equal to others already in R are not allowed to enter under any condition.

g. *Stopping criteria.* The scatter search is halted if: (i) the incumbent solution has value equal to lower bound L; or (ii) no reference set update occurs at Step 4.; or (iii) Step 4. has been executed 10 times.

2.5 The enumeration algorithm

The results of the previous sections have been embedded into a depth-first branch-and-bound algorithm, derived from that developed by Dell'Amico and Martello [41] for $P||C_{\max}$. At level j of the branch-decision tree, let M_j be the subset of processors i satisfying $k(i) < k$ and $C(i) + p_j < z$: $|M_j|$ nodes are then generated, by assigning job j to the processors in M_j. In order to avoid the generation of equivalent solutions, only processors with different $C(i)$ or $k(i)$ value are considered during the branching phase.

At the root node, the algorithm computes the overall lower bound (see Section 2.2)

$$L = \max\{\overline{L}_2, L_3, L_3^k, L_{BKK}, L_{HS}\}$$

where $\overline{L}_2$, according to the specific instance, denotes L_2 or $\widetilde{L}_2$ or $\widetilde{L}'_2$. In addition, heuristics LPT, MS_k and $MS2_k$ (with possible improvement of L, see Section 2.3.2) are executed, followed by the Scatter Search of Section 2.4.

At each node other than the root, three lower bounds are computed, in sequence, for the current instance: A modified continuous bound LC, $L3$ and $L3^k$. Since at any intermediate node a partial solution has been already defined, the lower bounding procedure has to be applied to the remaining sub-instance, but taking into account the fixed decisions as follows.

Lower bound $L3$ is locally computed as in [41]. For $L3^k$, let $\hat{\imath} = \arg\min\{C(h)\}$. We first remove from the instance all the assigned jobs. Then we add, for each processor, a dummy job j with processing time $\widetilde{p}_j = C(i) - C(\hat{\imath})$ (excluding dummy jobs with $\widetilde{p}_j = 0$). The cardinality limit k is then decreased by the minimum number of jobs completely executed on any processor in time interval $[0, C(\hat{\imath})]$. (In order to minimize the resulting k value, it is convenient to sort,

for each processor, the scheduled jobs according to non-decreasing processing time.) The bound of the node is then given by $C(\hat{\imath})$ plus the lower bound computed on the instance induced by the dummy and the unassigned jobs.

For the modified continuous bound LC, the fixed decisions are taken into account by: (i) excluding the processors with $k(i) = k$ or $C(i) + p_n \geq z$; (ii) assigning to all processors i with $k(i) = k - 1$ the longest unassigned job j such that $C(i) + p_j < z$, and excluding these processors and jobs; (iii) computing the continuous bound $C = \lceil \sum_{j \in J} p_j / \overline{m} \rceil$, where J is the set of unassigned jobs and $\overline{m}$ is the number of non-excluded processors, and setting $LC = \max(C, \max\{C(i) : k(i) = k \text{ or } C(i) + p_n \geq z\})$

The enumeration algorithm also includes dominance considerations. Three dominance criteria were introduced in [41] for $P||C_{\max}$. One of these (Criterion 1: If $p_j = p_{j+1}$ and j is currently assigned to processor h, at level $j+1$ only processors i satisfying $C(i) \geq C(h) - p_j$ are considered for the assignment of job $j+1$) directly applies to $P|\#\leq k|C_{\max}$. The other two were adapted to the cardinality constraint, and are as follows. Let I denote the current set of processors with $k(i) < k$:

> Criterion 2: At level j, let $\overline{n} = n - j + 1$ be the number of unassigned jobs. If $\overline{n} < |I|$, only the $\overline{n}$ processors of I with smallest $C(i)$ must be considered for the assignment of job j.
>
> Criterion 3: At level $n-2$, let $i_{\min}$ (resp. i_{smin}) be the processor of K with minimum (resp. second minimum, if any) $C(i)$. If $k(i_{\min}) = k - 1$ or $|I| = 1$, the optimal completion of the current schedule is the solution produced by the LPT_k rule. Otherwise it is the best between the LPT_k solution and that obtained by assigning job $n-2$ to i_{smin} and jobs $n-1$ and n to $i_{\min}$.

We finally describe a fathoming criterion adopted, for the KPP instances, at each decision-node, where job j is assigned to processor i. If, after such assignment, we have $0 < k(i) \leq k-2$, we can consider the minimum possible addition $s(i)$ to the processing time of i: $s(i) = \sum_{h=n-(k-k(i))+1}^{n} p_h$ (total processing time of the smallest $k - k(i)$ jobs). If $C(i) + s(i) \geq z$, the node can immediately be fathomed. If instead $C(i) + s(i) < z$, consider the previous job $r(i) = n - (k - k(i))$. If $C(i) + s(i) - p_n + p_{r(i)} \geq z$, we know that job n has to be assigned to i. Hence, we can fathom the node if there exists a processors $q \neq i$ for which the minimum possible makespan, $C(q) + s(q) - p_n + p_{r(q)}$, is no less than z.

2.6 Computational experiments

The algorithms of the previous sections have been coded in C++ and experimentally tested, on a DELL Dimension 8250 with Intel Pentium IV at 2.4 GHz running under a Windows 2000 operating system, both on random instances and on real world instances.

2.6.1 Random instances

We used fifteen classes of randomly generated instances. The first nine classes were already adopted in the computational experiments in [38]:

Classes 1, 2 and 3: uniform distribution with p_j in $[10, 1000]$, $[200, 1000]$ and $[500, 1000]$, respectively;

Classes 4, 5 and 6: exponential distribution of average value $\mu = 25$, $\mu = 50$ and $\mu = 100$,

respectively, by disregarding non-positive values;

Classes 7, 8 and 9: normal distribution of average value $\mu = 100$, and standard deviation $\sigma = 33$, $\sigma = 66$ and $\sigma = 100$, respectively, by disregarding non-positive values

In order to investigate more challenging problems, six additional classes were adopted:

Classes 10, 11 and 12: KPP instances with uniform distribution and p_j in $[500, 10000]$, $[1000, 10000]$ and $[1500, 10000]$, respectively;

Classes 13, 14 and 15: "perfect packing" KPP instances with $z = \sum_{j=1}^{n} p_j/m$, and $z = 1000$, $z = 5000$ and $z = 10000$, respectively. These were obtained by uniformly randomly splitting, for each processor, the segment $[1, z]$ into k segments.

For Classes 1–9, the code was tested with the values $m = (3, 4, 5, 10, 20, 40, 50)$, $n = (10, 25, 50, 100, 200, 400)$ and $k = (3, 4, 5, 10, 20, 40, 50)$. For Classes 10–15, the values were $m = (8, 10, 13, 15, 18, 20, 25, 30)$ and $k = (3, 4, 5, 10, 15, 20, 25)$, with $n = mk \leq 400$. In order to avoid trivial problems, we considered only instances satisfying $n > 2m$, $n/m \leq k \leq n/2$ and $mk \leq 4n$. For each triple (n, m, k) 10 instances were generated, hence, in total, 720 instances for each class 1–9, and 490 instances for each class 10–15.

Table 2.1 presents the overall performance of all algorithms over each class. We evaluated the separate performances of the three initial heuristics (LPT_k, MS_k and $MS2_k$), the performance of the scatter search (*Scatter 0* and *Scatter 1*) and that of the enumeration algorithm ($B\&B$). The scatter search was evaluated both when executed from scratch, i.e., with the first reference set containing only random solutions (*Scatter 0*) and when normally executed, i.e., by receiving in input the best solution found by the initial heuristics (*Scatter 1*). The enumeration algorithm was executed with a limit of 10 000 backtrackings. The execution time was not a problem: the maximum CPU time required for the complete execution (lower bounds, initial heuristics, *Scatter 1* and $B\&B$) on any instance was less than four minutes. Let z_A be the value of the solution found by an algorithm A, and z the best solution value obtained. For each algorithm A and for each class we give:

- #best = number of times in which $z_A = z$;
- #opt = number of times in which $z_A = z$ and z was proved to be optimal;
- #missed = number of times in which $z_A > z$ and z was proved to be optimal;
- %gap = average percentage gap. For each instance, the gap was computed as $100(z_A - z)/z$ if z was proved to be optimal, or as $100(z_A - L)/L$ otherwise.

The performance of the scatter search is very good, especially when executed after the initial heuristics. The quality of the approach is also proved by the fact that its performance deteriorates very little if it is executed from scratch.

The results obtained are also represented in Figures 2.1–2.5, where white (resp. dashed) bars represent the average percentage values of *#best* (resp. *#opt*) for groups of three similar classes. Exponentially distributed processing times (Classes 4–6) produce the easiest problems, whereas Classes 1–3 and 7–9 are more difficult. The high percentage of optimal solutions found proves however that both the heuristics and the lower bound perform very well for Classes 1–9. The KPP instances (Classes 10–15) are the hardest ones. The number of proved optimal solutions is small, especially for Classes 10–12, for which the lower bound has a poor performance.

The simple heuristics LPT_k, MS_k and $MS2_k$ have acceptable performances for Classes 1–9, but give very bad results for the KPP instances. The total number of best solutions found by $MS2_k$ is larger than that found by LPT_k, but its average percentage error is always much higher. This can be explained by observing that $MS2_k$, at each leaf, tries to obtain a solution of value equal to the lower bound: when no leaf succeeds in assigning all items, the completion produced by LPT_k can be quite bad. The Scatter Search, even if executed from scratch, always outperforms the other heuristics, both with respect to the number of optimal solutions and to the percentage gap.

Tables 2.2 – 2.6 present in detail the results of the overall algorithm for all classes. For each value of n, m and k, column *opt* gives the number of optimal solutions found (out of ten instances), column $\%g$ the average percentage gap, column $\%g_M$ the maximum percentage gap, and columns t and t_M the average and maximum elapsed CPU time. In addition, for each value of m, there is a row summarizing the average results. The final row of each table gives the overall average on all the instances of the class.

We can observe that larger values of n or k generally give easier instances. Indeed, the initial heuristics tend to produce much better solutions in these cases. No immediate relation is observed instead between the value of m and the difficulty of the instances. Increasing the values of the processing times gives in general easier instances for Classes 1–12, but harder instances for the "perfect packing" KPP case.

Worth is noting that the average and maximum gap are very small for all instances, and in the great majority of cases they are below 1%.

2.6.2 Real world instances

As mentioned in the Introduction, $P|\#\leq k|C_{\max}$ finds applications in robotized assembly lines. Hillier and Brandeau [89] studied an operation assignment problem arising from a Printed Circuit Board (PCB) assembly process inspired by an application at Hewlett-Packard (HP). They experimented their Lagrangian heuristic on four data sets based on real instances provided by HP. Each data set is characterized by

b = number of different board types;
c = number of component types;
d_i = demand of boards of type i ($i = 1, \ldots, b$);
ν_{ij} = number of components of type j to be placed on a board of type i;
p = processing time for placing a component (of any type);
$\overline{k}$ = maximum number of component types that can be assigned to any processor.

For each PCB data set, we constructed two $P|\#\leq k|C_{\max}$ data sets as follows:

PCB_1: for each board type i ($i = 1, \ldots, b$) and component type j ($j = 1, \ldots, c$) we define a job with processing time $p\nu_{ij}d_i$;

PCB_2: for each component type j ($j = 1, \ldots, c$) we define a job with processing time $p\sum_{i=1}^{b}(\nu_{ij}d_i)$.

Worth is mentioning that the values of n obtained in this way are considerably higher than those tested on random instances (see Table 2.7). For each data set, we solved the three instances obtained by setting $k = \lceil n/m \rceil$, i.e., the minimum value for which a feasible solution exists, and $m = (5, 10, 20)$. (We also attempted higher values of k without observing relevant variations.)

Table 2.7 presents the results obtained on the 24 resulting instances by all the considered algorithms. For each value of n, m and k, column L gives the best lower bound value, column z the best solution value obtained, column *opt* the value 1 (resp. 0) if the solution of value z was (resp. was not) proved to be optimal. The six next columns give, for each algorithm, the percentage gap between the solution value found and z (if z is optimal) or L (otherwise). The last column gives the elapsed CPU time of the overall algorithm. For each data set, the final row summarizes the average results. The overall average is given in the last row.

The table shows that the simplest heuristic, LPT_k, has a very good performance, by far dominating that of MS_k and $MS2_k$. The performance of the scatter search is excellent, outperforming the percentage error of LPT_k by two orders of magnitude, both if executed from scratch and starting from the best heuristic solution. The branch-and-bound algorithm could never improve a scatter search solution, indicating that, for these instances, it is difficult to prove the optimality of a solution. Allowing more than 10 000 backtrackings did not produce improvements.

In conclusion, the overall performance of the proposed scatter search algorithm is very satisfactory both on random instances and real world data sets.

Table 2.1: Overall performance of the algorithms for Classes 1–15.

Class		LPT_k	MS_k	$MS2_k$	*Scatter 0*	*Scatter 1*	*B&B*
	#best	81	181	303	651	720	720
1	#opt	80	181	301	587	646	646
	#missed	566	465	345	59	0	0
	%gap	1.655	8.246	6.073	0.052	0.047	0.047
	#best	46	248	345	662	720	720
2	#opt	44	246	344	561	611	611
	#missed	567	365	267	50	0	0
	%gap	2.668	8.752	6.275	0.093	0.090	0.090
	#best	64	263	372	700	715	720
3	#opt	64	263	371	657	671	676
	#missed	612	413	305	19	5	0
	%gap	2.965	5.500	3.747	0.161	0.160	0.160
	#best	572	290	326	720	720	720
4	#opt	572	290	326	720	720	720
	#missed	148	430	394	0	0	0
	%gap	0.235	3.810	3.108	0.000	0.000	0.000
	#best	474	300	335	720	720	720
5	#opt	474	300	335	720	720	720
	#missed	246	420	385	0	0	0
	%gap	0.261	3.597	2.999	0.000	0.000	0.000
	#best	381	306	351	720	720	720
6	#opt	381	306	351	718	718	718
	#missed	337	412	367	0	0	0
	%gap	0.276	3.587	2.887	0.001	0.001	0.001
	#best	37	297	359	719	720	720
7	#opt	37	297	359	637	638	638
	#missed	601	341	279	1	0	0
	%gap	3.618	8.018	5.756	0.093	0.093	0.093
	#best	86	258	334	712	720	720
8	#opt	86	258	333	661	667	667
	#missed	581	409	334	6	0	0
	%gap	2.250	6.842	4.712	0.049	0.045	0.045
	#best	140	267	333	715	720	720
9	#opt	140	267	333	684	687	687
	#missed	547	420	354	3	0	0
	%gap	1.404	5.605	4.165	0.025	0.023	0.023
	#best	1	0	0	400	490	490
10	#opt	1	0	0	197	223	223
	#missed	222	223	223	26	0	0
	%gap	1.859	12.050	8.835	0.045	0.043	0.043
	#best	1	1	0	418	489	490
11	#opt	1	1	0	217	231	231
	#missed	230	230	231	14	0	0
	%gap	1.682	12.467	9.522	0.041	0.039	0.039
	#best	1	0	0	412	487	490
12	#opt	1	0	0	224	245	246
	#missed	245	246	246	22	1	0
	%gap	1.520	12.815	9.843	0.037	0.036	0.035
	#best	17	1	2	478	483	490
13	#opt	17	1	2	420	425	432
	#missed	415	431	430	12	7	0
	%gap	1.421	13.529	12.554	0.018	0.017	0.015
	#best	0	1	2	470	487	490
14	#opt	0	1	2	375	380	380
	#missed	380	379	378	5	0	0
	%gap	1.400	13.672	12.119	0.015	0.014	0.014
	#best	0	1	2	462	489	490
15	#opt	0	1	2	343	353	353
	#missed	353	352	351	10	0	0
	%gap	1.400	13.603	11.790	0.015	0.014	0.014

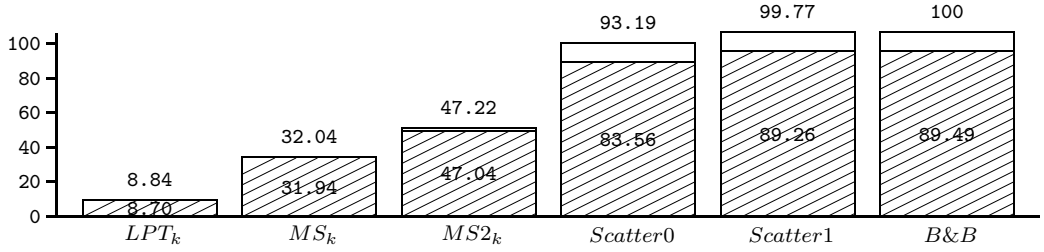

Figure 2.1: Percentage of optimal solutions for Classes 1–3

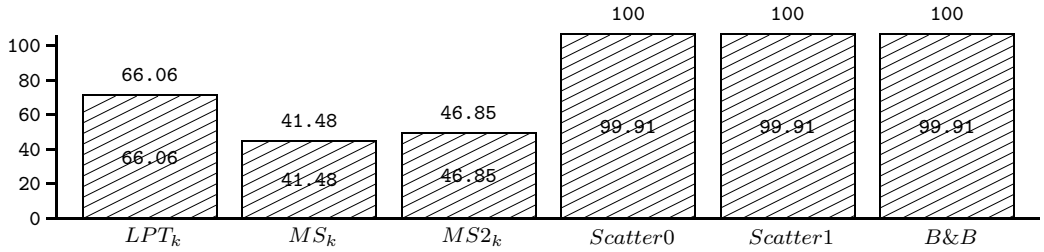

Figure 2.2: Percentage of optimal solutions for Classes 4–6

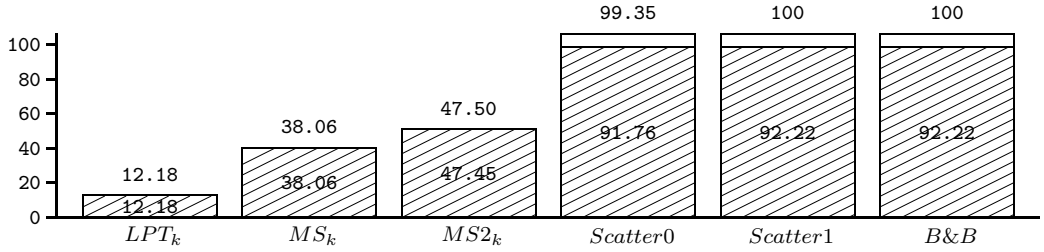

Figure 2.3: Percentage of optimal solutions for Classes 7–9

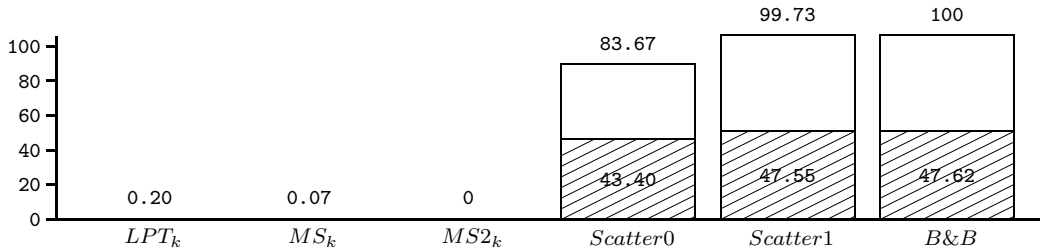

Figure 2.4: Percentage of optimal solutions for Classes 10–12

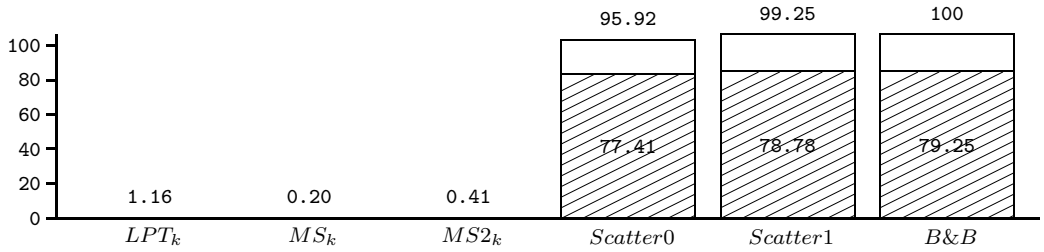

Figure 2.5: Percentage of optimal solutions for Classes 13–15

Table 2.2: Results for Classes 1–3 (uniform distribution).

			$p_j \in [10, 1000]$					$p_j \in [200, 1000]$					$p_j \in [500, 1000]$				
n	m	k	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M
10	3	4	10	0.000	0.000	0.03	0.06	10	0.000	0.000	0.04	0.04	10	0.000	0.000	0.02	0.04
10	3	5	10	0.000	0.000	0.02	0.06	10	0.000	0.000	0.03	0.05	10	0.000	0.000	0.01	0.04
25	3	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	3	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	3	40	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.04	0.10	10	0.000	0.000	0.00	0.00
100	3	50	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
average			10.0	0.000	0.000	0.01	0.02	10.0	0.000	0.000	0.02	0.03	10.0	0.000	0.000	0.01	0.01
10	4	3	10	0.000	0.000	0.01	0.06	10	0.000	0.000	0.03	0.05	10	0.000	0.000	0.04	0.05
10	4	4	10	0.000	0.000	0.03	0.07	10	0.000	0.000	0.04	0.05	10	0.000	0.000	0.04	0.05
10	4	5	10	0.000	0.000	0.04	0.09	10	0.000	0.000	0.03	0.06	10	0.000	0.000	0.04	0.07
25	4	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	4	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	4	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	4	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	4	50	10	0.000	0.000	0.02	0.05	10	0.000	0.000	0.02	0.04	10	0.000	0.000	0.01	0.03
average			10.0	0.000	0.000	0.01	0.03	10.0	0.000	0.000	0.01	0.02	10.0	0.000	0.000	0.02	0.03
10	5	3	10	0.000	0.000	0.00	0.04	10	0.000	0.000	0.01	0.04	10	0.000	0.000	0.00	0.00
10	5	4	10	0.000	0.000	0.00	0.04	10	0.000	0.000	0.00	0.05	10	0.000	0.000	0.00	0.00
10	5	5	10	0.000	0.000	0.01	0.04	10	0.000	0.000	0.00	0.04	10	0.000	0.000	0.00	0.00
25	5	5	8	0.008	0.039	0.11	0.58	7	0.011	0.043	0.19	0.55	9	0.003	0.026	0.05	0.46
25	5	10	9	0.004	0.039	0.07	0.64	10	0.000	0.000	0.00	0.02	10	0.000	0.000	0.00	0.02
50	5	10	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.00	0.01
50	5	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	5	20	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.01	0.01
100	5	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	5	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	5	40	10	0.000	0.000	0.02	0.07	10	0.000	0.000	0.02	0.05	10	0.000	0.000	0.01	0.03
200	5	50	10	0.000	0.000	0.02	0.03	10	0.000	0.000	0.02	0.02	10	0.000	0.000	0.00	0.01
average			9.8	0.001	0.006	0.02	0.12	9.8	0.001	0.004	0.02	0.07	9.9	0.000	0.002	0.01	0.05
25	10	3	10	0.000	0.000	0.36	0.68	8	0.169	1.617	0.40	0.87	10	0.000	0.000	0.31	0.43
25	10	4	10	0.000	0.000	0.31	0.73	9	0.202	2.021	0.55	0.85	10	0.000	0.000	0.39	0.57
25	10	5	10	0.000	0.000	0.19	0.66	7	0.806	3.068	0.60	0.97	9	0.097	0.966	0.35	0.60
25	10	10	10	0.000	0.000	0.44	0.77	8	0.490	2.828	0.63	0.99	10	0.000	0.000	0.39	0.56
50	10	5	10	0.000	0.000	0.02	0.06	10	0.000	0.000	0.02	0.07	10	0.000	0.000	0.01	0.02
50	10	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.00	9	0.003	0.027	0.22	2.15
50	10	20	9	0.004	0.043	0.30	2.94	9	0.003	0.032	0.22	2.20	10	0.000	0.000	0.04	0.39
100	10	10	10	0.000	0.000	0.03	0.07	10	0.000	0.000	0.02	0.03	10	0.000	0.000	0.01	0.02
100	10	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	10	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	10	20	10	0.000	0.000	0.03	0.08	10	0.000	0.000	0.03	0.06	10	0.000	0.000	0.02	0.03
200	10	40	10	0.000	0.000	0.03	0.06	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	10	50	10	0.000	0.000	0.01	0.05	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
400	10	40	10	0.000	0.000	0.07	0.09	10	0.000	0.000	0.06	0.08	10	0.000	0.000	0.04	0.05
400	10	50	10	0.000	0.000	0.05	0.06	10	0.000	0.000	0.04	0.05	10	0.000	0.000	0.03	0.05
average			9.9	0.000	0.003	0.12	0.42	9.4	0.111	0.638	0.17	0.41	9.9	0.007	0.066	0.12	0.33
50	20	3	6	0.251	0.746	2.17	5.36	0	0.637	1.105	4.56	5.52	7	0.653	2.853	4.05	6.16
50	20	4	4	0.383	1.111	2.41	4.54	0	0.685	1.697	4.51	5.22	9	0.020	0.199	4.46	6.63
50	20	5	3	0.528	1.325	3.28	5.33	0	0.706	1.649	4.72	7.06	8	0.873	4.836	5.21	8.21
50	20	10	2	0.448	0.819	3.88	5.71	0	0.578	1.023	4.93	6.48	9	0.407	4.073	4.66	6.87
100	20	5	10	0.000	0.000	0.11	0.38	10	0.000	0.000	0.07	0.23	10	0.000	0.000	0.06	0.22
100	20	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	5	0.013	0.027	6.31	13.49
100	20	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	9	0.003	0.026	1.19	11.16
200	20	10	10	0.000	0.000	0.09	0.16	10	0.000	0.000	0.07	0.12	10	0.000	0.000	0.04	0.08
200	20	20	8	0.004	0.020	5.74	28.76	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	20	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
400	20	20	10	0.000	0.000	0.11	0.16	10	0.000	0.000	0.09	0.14	10	0.000	0.000	0.05	0.08
400	20	40	10	0.000	0.000	0.14	0.27	10	0.000	0.000	0.06	0.15	10	0.000	0.000	0.00	0.00
400	20	50	10	0.000	0.000	0.11	0.19	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
average			7.9	0.124	0.309	1.39	3.91	6.9	0.201	0.421	1.46	1.92	9.0	0.151	0.924	2.00	4.07
100	40	3	5	0.207	0.516	15.44	34.18	0	0.408	0.747	34.29	43.44	6	1.364	3.907	46.90	55.68
100	40	4	1	0.460	0.732	26.71	33.05	0	0.385	0.543	33.77	42.56	3	2.998	5.260	53.44	72.02
100	40	5	1	0.441	0.801	29.86	37.14	0	0.369	0.707	30.43	34.62	3	2.951	5.407	58.03	66.20
100	40	10	1	0.476	0.701	29.01	43.52	0	0.387	0.492	35.87	44.61	4	2.126	4.914	62.08	77.87
200	40	5	10	0.000	0.000	0.80	2.86	10	0.000	0.000	1.91	5.24	9	0.003	0.027	12.03	114.66
200	40	10	9	0.004	0.043	4.92	49.20	10	0.000	0.000	0.00	0.00	9	0.003	0.027	7.05	68.22
200	40	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	9	0.003	0.027	8.37	63.04
400	40	10	10	0.000	0.000	0.40	1.13	10	0.000	0.000	0.24	0.86	10	0.000	0.000	0.14	0.39
400	40	20	10	0.000	0.000	4.37	34.42	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
400	40	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
average			6.7	0.159	0.279	11.15	23.55	6.0	0.155	0.249	13.65	17.14	7.3	0.945	1.957	24.80	51.81
100	50	3	9	0.057	0.569	3.91	38.80	7	0.226	1.084	10.02	38.40	10	0.000	0.000	0.01	0.01
100	50	4	8	0.076	0.397	6.67	34.88	7	0.157	1.007	12.18	33.64	10	0.000	0.000	0.00	0.01
100	50	5	9	0.030	0.302	5.68	30.07	3	0.296	1.092	21.24	37.24	10	0.000	0.000	0.01	0.01
200	50	4	10	0.000	0.000	0.69	5.39	10	0.000	0.000	0.20	0.62	10	0.000	0.000	0.25	1.68
200	50	5	5	0.025	0.052	39.54	88.69	6	0.017	0.043	30.91	95.25	9	0.003	0.034	10.88	105.53
200	50	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.02	9	0.003	0.034	12.63	126.31
400	50	10	10	0.000	0.000	1.62	11.74	9	0.002	0.021	10.34	98.58	10	0.000	0.000	0.01	0.03
400	50	20	8	0.005	0.026	26.23	129.79	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.01	0.03
average			8.6	0.024	0.168	10.54	42.42	7.8	0.087	0.406	10.61	37.97	9.8	0.001	0.008	2.98	29.20
overall average			9.0	0.047	0.115	3.00	8.80	8.5	0.091	0.289	3.38	7.05	9.4	0.160	0.454	4.17	11.25

Table 2.3: Results for Classes 4–6 (exponential distribution).

			$\mu = 25$					$\mu = 50$					$\mu = 100$				
n	m	k	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M
10	3	4	10	0.000	0.000	0.02	0.06	10	0.000	0.000	0.01	0.05	10	0.000	0.000	0.02	0.06
10	3	5	10	0.000	0.000	0.01	0.07	10	0.000	0.000	0.02	0.05	10	0.000	0.000	0.02	0.07
25	3	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	3	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	3	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
100	3	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
average			10.0	0.000	0.000	0.00	0.02	10.0	0.000	0.000	0.00	0.02	10.0	0.000	0.000	0.01	0.02
10	4	3	10	0.000	0.000	0.01	0.06	10	0.000	0.000	0.00	0.02	10	0.000	0.000	0.00	0.03
10	4	4	10	0.000	0.000	0.01	0.06	10	0.000	0.000	0.01	0.06	10	0.000	0.000	0.01	0.06
10	4	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.01	0.04	10	0.000	0.000	0.01	0.04
25	4	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	4	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
100	4	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
100	4	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
200	4	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
average			10.0	0.000	0.000	0.00	0.02	10.0	0.000	0.000	0.00	0.02	10.0	0.000	0.000	0.00	0.02
10	5	3	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
10	5	4	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
10	5	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
25	5	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.03	0.34	9	0.014	0.136	0.11	0.66
25	5	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	5	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
50	5	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
100	5	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
100	5	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.05
100	5	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	5	40	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
200	5	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
average			10.0	0.000	0.000	0.00	0.00	10.0	0.000	0.000	0.00	0.03	9.9	0.001	0.011	0.01	0.06
25	10	3	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
25	10	4	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.06	0.57
25	10	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.06	0.56	10	0.000	0.000	0.05	0.54
25	10	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	10	5	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.02
50	10	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
50	10	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	10	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.02
100	10	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
100	10	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	10	20	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.02
200	10	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
200	10	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
400	10	40	10	0.000	0.000	0.00	0.02	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.01	0.02
400	10	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.02
average			10.0	0.000	0.000	0.00	0.00	10.0	0.000	0.000	0.01	0.04	10.0	0.000	0.000	0.01	0.08
50	20	3	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	20	4	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	20	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	20	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	20	5	10	0.000	0.000	0.02	0.05	10	0.000	0.000	0.05	0.26	10	0.000	0.000	0.10	0.73
100	20	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.06
100	20	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	20	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.02	0.16	10	0.000	0.000	0.02	0.02
200	20	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.02
200	20	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.02
400	20	20	10	0.000	0.000	0.02	0.04	10	0.000	0.000	0.03	0.04	10	0.000	0.000	0.03	0.04
400	20	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.02	10	0.000	0.000	0.02	0.03
400	20	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.03	0.04
average			10.0	0.000	0.000	0.00	0.01	10.0	0.000	0.000	0.01	0.04	10.0	0.000	0.000	0.02	0.07
100	40	3	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	40	4	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	40	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	40	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	40	5	10	0.000	0.000	0.15	1.46	10	0.000	0.000	0.23	1.31	10	0.000	0.000	5.26	52.62
200	40	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.00	9	0.024	0.236	3.54	35.36
200	40	20	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
400	40	10	10	0.000	0.000	0.08	0.19	10	0.000	0.000	0.21	0.89	10	0.000	0.000	0.67	4.07
400	40	20	10	0.000	0.000	0.01	0.04	10	0.000	0.000	0.03	0.07	10	0.000	0.000	0.04	0.14
400	40	40	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.02	0.19	10	0.000	0.000	0.03	0.18
average			10.0	0.000	0.000	0.02	0.17	10.0	0.000	0.000	0.05	0.25	9.9	0.002	0.024	0.95	9.24
100	50	3	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	50	4	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	50	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	50	4	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	50	5	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	50	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.00
400	50	10	10	0.000	0.000	0.01	0.05	10	0.000	0.000	0.01	0.08	10	0.000	0.000	0.01	0.03
400	50	20	10	0.000	0.000	0.00	0.04	10	0.000	0.000	0.01	0.05	10	0.000	0.000	0.30	1.71
average			10.0	0.000	0.000	0.00	0.01	10.0	0.000	0.000	0.00	0.02	10.0	0.000	0.000	0.04	0.22
overall average			10.0	0.000	0.000	0.00	0.03	10.0	0.000	0.000	0.01	0.06	10.0	0.001	0.005	0.14	1.35

Table 2.4: Results for Classes 7–9 (normal distribution).

			$\mu = 100$, $\sigma = 33$					$\mu = 100$, $\sigma = 66$					$\mu = 100$, $\sigma = 100$				
n	m	k	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M
10	3	4	10	0.000	0.000	0.04	0.07	10	0.000	0.000	0.02	0.04	10	0.000	0.000	0.03	0.05
10	3	5	10	0.000	0.000	0.04	0.06	10	0.000	0.000	0.02	0.06	10	0.000	0.000	0.03	0.05
25	3	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	3	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
100	3	40	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
100	3	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
average			10.0	0.000	0.000	0.01	0.02	10.0	0.000	0.000	0.01	0.02	10.0	0.000	0.000	0.01	0.02
10	4	3	10	0.000	0.000	0.03	0.07	10	0.000	0.000	0.03	0.06	10	0.000	0.000	0.01	0.04
10	4	4	10	0.000	0.000	0.04	0.07	10	0.000	0.000	0.05	0.09	10	0.000	0.000	0.04	0.08
10	4	5	10	0.000	0.000	0.04	0.07	10	0.000	0.000	0.02	0.07	10	0.000	0.000	0.02	0.08
25	4	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	4	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	4	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
100	4	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	4	50	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.02
average			10.0	0.000	0.000	0.02	0.03	10.0	0.000	0.000	0.01	0.03	10.0	0.000	0.000	0.01	0.03
10	5	3	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.01	0.04	10	0.000	0.000	0.00	0.03
10	5	4	10	0.000	0.000	0.01	0.06	10	0.000	0.000	0.00	0.04	10	0.000	0.000	0.00	0.00
10	5	5	10	0.000	0.000	0.01	0.07	10	0.000	0.000	0.01	0.10	10	0.000	0.000	0.00	0.04
25	5	5	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
25	5	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	5	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
50	5	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	5	20	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.02
100	5	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	5	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	5	40	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.01	0.02
200	5	50	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.00	0.01
average			10.0	0.000	0.000	0.00	0.02	10.0	0.000	0.000	0.00	0.02	10.0	0.000	0.000	0.00	0.01
25	10	3	10	0.000	0.000	0.48	0.73	10	0.000	0.000	0.41	0.58	10	0.000	0.000	0.23	0.53
25	10	4	6	0.922	4.082	0.70	1.08	9	0.394	3.943	0.49	1.03	10	0.000	0.000	0.30	0.64
25	10	5	8	0.360	1.961	0.65	1.00	8	0.425	2.545	0.52	0.93	10	0.000	0.000	0.35	0.89
25	10	10	6	0.841	2.672	0.76	1.20	10	0.000	0.000	0.53	0.71	10	0.000	0.000	0.51	0.94
50	10	5	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.01
50	10	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
50	10	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	10	10	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.01	0.02
100	10	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
100	10	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	10	20	10	0.000	0.000	0.02	0.03	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.01	0.01
200	10	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.01	0.01
200	10	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.01	0.01
400	10	40	10	0.000	0.000	0.03	0.05	10	0.000	0.000	0.02	0.03	10	0.000	0.000	0.02	0.03
400	10	50	10	0.000	0.000	0.03	0.06	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.01	0.02
average			9.3	0.142	0.581	0.18	0.28	9.8	0.055	0.433	0.13	0.22	10.0	0.000	0.000	0.10	0.21
50	20	3	3	0.444	1.172	3.91	5.02	3	0.371	1.056	3.20	4.51	5	0.303	0.880	2.20	5.65
50	20	4	2	0.626	2.335	4.92	9.72	5	0.221	0.787	2.37	5.15	6	0.296	1.079	2.31	5.05
50	20	5	3	0.317	0.787	4.03	5.93	2	0.514	0.823	3.83	6.00	7	0.217	0.901	2.07	5.58
50	20	10	4	0.323	0.797	3.95	6.51	2	0.450	0.844	3.97	5.42	2	0.462	0.990	3.91	5.39
100	20	5	10	0.000	0.000	0.02	0.05	10	0.000	0.000	0.02	0.03	10	0.000	0.000	0.02	0.07
100	20	10	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.01
100	20	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	20	10	10	0.000	0.000	0.02	0.03	10	0.000	0.000	0.03	0.03	10	0.000	0.000	0.03	0.06
200	20	20	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.02	0.03	10	0.000	0.000	0.02	0.04
200	20	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
400	20	20	10	0.000	0.000	0.04	0.06	10	0.000	0.000	0.04	0.05	10	0.000	0.000	0.04	0.05
400	20	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.03	0.05	10	0.000	0.000	0.03	0.05
400	20	50	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.03	0.05	10	0.000	0.000	0.03	0.04
average			7.8	0.132	0.392	1.30	2.10	7.8	0.120	0.270	1.04	1.64	8.5	0.098	0.296	0.82	1.69
100	40	3	0	0.479	0.794	25.09	32.19	3	0.255	0.377	16.85	28.36	6	0.154	0.631	9.90	25.87
100	40	4	3	0.401	1.575	22.68	34.66	3	0.259	0.380	17.34	25.62	8	0.091	0.625	4.90	24.54
100	40	5	4	0.323	1.181	17.60	33.06	6	0.215	0.714	10.35	30.18	9	0.033	0.326	2.25	22.47
100	40	10	2	0.543	1.460	23.69	35.89	7	0.103	0.352	8.59	29.35	10	0.000	0.000	0.00	0.00
200	40	5	10	0.000	0.000	0.04	0.05	10	0.000	0.000	0.78	2.24	9	0.016	0.164	22.83	206.73
200	40	10	10	0.000	0.000	0.03	0.32	9	0.019	0.192	3.65	36.43	9	0.017	0.168	3.84	35.90
200	40	20	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
400	40	10	10	0.000	0.000	0.08	0.13	10	0.000	0.000	0.08	0.09	10	0.000	0.000	0.10	0.30
400	40	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.10	0.32	10	0.000	0.000	0.09	0.26
400	40	40	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.03	0.16
average			6.9	0.175	0.501	8.92	13.63	7.8	0.085	0.202	5.78	15.26	9.1	0.031	0.191	4.39	31.62
100	50	3	7	0.284	1.429	11.66	37.78	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
100	50	4	7	0.337	1.463	14.17	31.74	9	0.083	0.833	3.17	31.53	10	0.000	0.000	0.00	0.00
100	50	5	2	0.546	1.508	22.45	36.60	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.00	0.00
200	50	4	10	0.000	0.000	0.03	0.04	10	0.000	0.000	0.05	0.11	10	0.000	0.000	0.18	1.21
200	50	5	10	0.000	0.000	0.02	0.06	10	0.000	0.000	0.50	4.30	7	0.059	0.205	18.99	51.35
200	50	10	10	0.000	0.000	0.87	8.64	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.11	0.55
400	50	10	10	0.000	0.000	0.06	0.07	10	0.000	0.000	0.23	1.66	10	0.000	0.000	0.09	0.18
400	50	20	10	0.000	0.000	0.00	0.00	10	0.000	0.000	0.05	0.09	10	0.000	0.000	0.07	0.11
average			8.3	0.146	0.550	6.16	14.37	9.9	0.010	0.104	0.50	4.71	9.6	0.007	0.026	2.43	6.68
overall average			8.8	0.094	0.322	2.20	3.94	9.3	0.046	0.178	1.08	2.99	9.6	0.023	0.083	1.05	5.49

Table 2.5: Results for Classes 10–12 (KPP instances).

			$p_j \in [500, 10000]$					$p_j \in [1000, 10000]$					$p_j \in [1500, 10000]$				
n	m	k	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M
24	8	3	10	0.000	0.000	0.35	0.60	10	0.000	0.000	0.32	0.60	10	0.000	0.000	0.31	0.60
32	8	4	0	0.071	0.100	1.23	1.53	0	0.056	0.079	1.21	1.36	0	0.063	0.116	1.19	1.30
40	8	5	0	0.032	0.053	1.38	1.56	0	0.025	0.035	1.36	1.53	0	0.025	0.037	1.35	1.50
80	8	10	0	0.004	0.007	2.98	3.38	0	0.003	0.005	3.04	3.35	1	0.003	0.005	2.66	3.37
120	8	15	2	0.001	0.001	5.06	6.62	5	0.001	0.001	2.92	6.47	2	0.001	0.001	4.92	6.32
160	8	20	7	0.000	0.001	4.08	9.81	7	0.000	0.001	3.24	9.81	9	0.000	0.001	1.45	8.83
200	8	25	10	0.000	0.000	1.86	12.71	8	0.000	0.001	3.18	14.69	8	0.000	0.001	2.69	12.34
average			4.1	0.015	0.023	2.42	5.17	4.3	0.012	0.018	2.18	5.40	4.3	0.013	0.023	2.08	4.90
30	10	3	10	0.000	0.000	0.73	1.42	10	0.000	0.000	0.68	1.39	10	0.000	0.000	0.72	1.34
40	10	4	0	0.076	0.127	1.77	1.97	0	0.063	0.095	1.82	2.05	0	0.063	0.081	1.68	1.95
50	10	5	0	0.034	0.051	2.37	2.91	0	0.026	0.038	2.28	2.90	0	0.024	0.038	2.17	2.50
100	10	10	0	0.005	0.006	5.90	6.99	0	0.004	0.006	5.74	6.73	0	0.004	0.006	5.42	6.71
150	10	15	2	0.001	0.003	9.78	13.60	4	0.001	0.001	7.22	11.91	3	0.001	0.002	8.10	13.76
200	10	20	3	0.001	0.001	12.86	20.12	8	0.000	0.001	3.98	20.02	8	0.000	0.001	4.47	20.23
250	10	25	8	0.000	0.001	6.45	31.02	8	0.000	0.001	7.95	25.12	9	0.000	0.001	6.03	29.90
average			3.3	0.017	0.027	5.69	11.15	4.3	0.014	0.020	4.24	10.02	4.3	0.013	0.018	4.08	10.91
39	13	3	2	0.566	1.698	2.06	2.65	2	0.515	1.539	2.06	2.74	2	0.469	1.394	2.11	2.67
52	13	4	0	0.080	0.111	3.22	3.61	0	0.062	0.087	3.06	3.45	0	0.062	0.087	3.10	3.49
65	13	5	0	0.032	0.053	4.46	4.96	1	0.022	0.042	4.00	4.73	1	0.016	0.033	3.75	4.70
130	13	10	1	0.003	0.006	11.70	15.20	3	0.003	0.006	10.12	17.28	3	0.002	0.004	9.02	14.11
195	13	15	5	0.001	0.003	11.05	25.37	5	0.001	0.001	12.93	27.90	7	0.001	0.002	6.86	26.72
260	13	20	5	0.000	0.001	21.90	46.38	8	0.000	0.001	8.56	41.63	7	0.000	0.001	12.74	44.61
325	13	25	10	0.000	0.000	4.98	21.78	8	0.000	0.001	13.41	64.62	9	0.000	0.001	6.51	57.53
average			3.3	0.098	0.267	8.48	17.13	3.9	0.086	0.239	7.73	23.19	4.1	0.079	0.217	6.30	21.98
45	15	3	2	0.208	0.526	2.67	3.81	2	0.190	0.475	2.70	3.84	2	0.169	0.435	2.57	3.75
60	15	4	0	0.070	0.097	4.55	5.49	0	0.065	0.083	4.70	5.13	0	0.057	0.083	4.49	4.98
75	15	5	2	0.023	0.043	4.80	6.77	2	0.023	0.044	5.09	6.99	1	0.019	0.035	5.19	6.23
150	15	10	3	0.003	0.006	12.50	18.95	4	0.003	0.006	11.48	21.30	4	0.003	0.005	10.79	21.02
225	15	15	8	0.000	0.001	8.29	40.95	6	0.001	0.002	13.26	37.53	8	0.000	0.001	7.39	37.36
300	15	20	7	0.000	0.001	18.24	63.17	8	0.000	0.001	16.57	56.16	7	0.000	0.001	22.26	61.41
375	15	25	8	0.000	0.001	25.81	96.75	9	0.000	0.001	19.81	128.31	7	0.000	0.001	32.30	125.63
average			4.3	0.044	0.096	10.98	33.70	4.4	0.040	0.087	10.52	37.04	4.1	0.035	0.080	12.14	37.20
54	18	3	0	0.245	0.367	4.73	6.08	0	0.207	0.333	5.02	5.79	0	0.196	0.302	5.09	6.17
72	18	4	0	0.070	0.096	7.86	9.89	0	0.063	0.086	7.42	8.85	0	0.060	0.082	7.24	9.11
90	18	5	1	0.020	0.039	9.18	12.00	0	0.019	0.035	9.96	11.97	1	0.018	0.036	9.06	12.06
180	18	10	2	0.003	0.006	22.51	34.70	8	0.001	0.004	6.97	34.71	7	0.001	0.003	10.05	33.12
270	18	15	10	0.000	0.000	1.01	1.60	5	0.001	0.001	28.64	63.28	7	0.000	0.001	20.61	73.49
360	18	20	9	0.000	0.001	12.34	112.36	8	0.000	0.001	23.25	115.74	9	0.000	0.001	12.10	110.95
average			3.7	0.056	0.085	9.61	29.44	3.5	0.048	0.077	13.54	40.06	4.0	0.046	0.071	10.69	40.82
60	20	3	0	0.213	0.277	6.63	7.39	0	0.193	0.262	6.65	7.36	0	0.173	0.226	6.59	7.28
80	20	4	0	0.060	0.086	9.80	12.40	0	0.058	0.069	9.75	11.23	0	0.049	0.078	9.80	13.25
100	20	5	2	0.017	0.043	11.40	16.13	5	0.011	0.039	7.60	14.98	3	0.011	0.036	9.01	15.32
200	20	10	3	0.003	0.006	30.94	52.35	4	0.002	0.006	25.35	57.29	8	0.001	0.004	9.43	46.82
300	20	15	8	0.000	0.001	19.18	99.00	8	0.000	0.001	17.51	86.76	10	0.000	0.000	1.34	2.25
400	20	20	10	0.000	0.000	1.76	2.58	7	0.000	0.001	48.84	164.88	10	0.000	0.000	1.32	2.07
average			3.8	0.049	0.069	13.29	31.64	4.0	0.044	0.063	19.28	57.08	5.2	0.039	0.057	6.25	14.50
75	25	3	0	0.177	0.231	11.50	14.18	0	0.158	0.208	11.41	14.06	0	0.151	0.170	11.30	13.65
100	25	4	1	0.040	0.072	15.79	23.74	0	0.038	0.073	16.02	25.71	0	0.028	0.052	14.71	21.94
125	25	5	3	0.011	0.033	18.21	28.65	2	0.013	0.033	21.07	28.20	6	0.006	0.030	12.77	28.03
250	25	10	8	0.000	0.002	17.50	86.61	9	0.001	0.005	9.42	76.18	8	0.000	0.002	16.98	91.37
375	25	15	9	0.000	0.001	20.96	189.89	9	0.000	0.001	18.52	165.76	10	0.000	0.000	1.89	2.79
average			4.2	0.046	0.068	16.79	68.61	4.0	0.042	0.064	15.29	61.98	4.8	0.037	0.051	11.53	31.56
90	30	3	0	0.169	0.196	19.32	22.02	0	0.156	0.187	19.41	22.02	0	0.137	0.174	19.03	21.71
120	30	4	0	0.027	0.050	25.06	29.65	0	0.028	0.055	23.56	27.98	2	0.024	0.054	20.90	29.07
150	30	5	6	0.005	0.023	20.33	47.16	6	0.004	0.019	18.89	55.34	8	0.004	0.025	13.50	55.06
300	30	10	9	0.000	0.004	18.62	152.93	10	0.000	0.000	1.56	2.60	9	0.000	0.002	15.86	141.61
average			3.8	0.050	0.068	20.83	62.94	4.0	0.047	0.065	15.86	26.99	4.8	0.041	0.064	17.32	61.86
overall average			3.8	0.046	0.090	10.16	29.21	4.1	0.041	0.081	10.40	31.23	4.4	0.038	0.074	8.18	25.76

Table 2.6: Results for Classes 13–15 (KPP perfect packing instances).

			$z = 1000$					$z = 5000$					$z = 10000$				
n	m	k	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M	opt	$\%g$	$\%g_M$	t	t_M
24	8	3	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.02	0.04
32	8	4	9	0.010	0.100	0.21	1.25	10	0.000	0.000	0.02	0.05	10	0.000	0.000	0.03	0.07
40	8	5	10	0.000	0.000	0.01	0.02	0	0.036	0.060	1.48	1.65	1	0.036	0.060	1.32	1.59
80	8	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.05	0.29	4	0.006	0.010	1.71	2.86
120	8	15	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.02	0.04
160	8	20	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.02	0.04
200	8	25	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.02	0.03
average			9.9	0.001	0.014	0.03	0.19	8.6	0.005	0.009	0.23	0.29	7.9	0.006	0.010	0.45	0.67
30	10	3	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.03	0.07
40	10	4	2	0.080	0.100	1.44	2.06	5	0.044	0.120	0.89	1.88	8	0.021	0.110	0.32	1.63
50	10	5	10	0.000	0.000	0.02	0.08	2	0.030	0.060	1.88	2.76	0	0.042	0.060	2.42	2.67
100	10	10	10	0.000	0.000	0.00	0.01	10	0.000	0.000	0.04	0.12	6	0.004	0.010	1.75	4.56
150	10	15	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.02	0.04	10	0.000	0.000	0.08	0.30
200	10	20	10	0.000	0.000	0.01	0.01	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.03	0.06
250	10	25	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.02	0.02	10	0.000	0.000	0.03	0.04
average			8.9	0.011	0.014	0.21	0.31	8.1	0.011	0.026	0.41	0.70	7.7	0.010	0.026	0.67	1.33
39	13	3	10	0.000	0.000	0.01	0.04	10	0.000	0.000	0.02	0.04	10	0.000	0.000	0.04	0.08
52	13	4	2	0.080	0.100	2.40	3.24	0	0.092	0.120	3.24	3.50	2	0.074	0.120	2.73	3.65
65	13	5	10	0.000	0.000	0.06	0.11	3	0.016	0.040	3.43	5.61	0	0.033	0.050	4.71	5.66
130	13	10	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.06	0.20	6	0.004	0.010	3.37	8.40
195	13	15	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.04	0.12	10	0.000	0.000	0.05	0.07
260	13	20	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.03	0.04	10	0.000	0.000	0.05	0.07
325	13	25	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.03	0.04	10	0.000	0.000	0.05	0.08
average			8.9	0.011	0.014	0.36	0.50	7.6	0.015	0.023	0.98	1.36	6.9	0.016	0.026	1.57	2.57
45	15	3	9	0.030	0.300	0.31	2.99	10	0.000	0.000	0.03	0.04	10	0.000	0.000	0.05	0.10
60	15	4	3	0.070	0.100	2.85	4.70	0	0.082	0.120	4.66	5.18	1	0.081	0.120	4.36	5.23
75	15	5	10	0.000	0.000	0.09	0.38	4	0.012	0.020	4.49	9.35	0	0.027	0.040	7.35	9.67
150	15	10	10	0.000	0.000	0.01	0.03	10	0.000	0.000	0.06	0.14	10	0.000	0.000	0.21	1.31
225	15	15	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.03	0.04	10	0.000	0.000	0.06	0.11
300	15	20	10	0.000	0.000	0.02	0.02	10	0.000	0.000	0.04	0.04	10	0.000	0.000	0.07	0.10
375	15	25	10	0.000	0.000	0.01	0.02	10	0.000	0.000	0.03	0.05	10	0.000	0.000	0.05	0.10
average			8.9	0.014	0.057	0.47	1.17	7.7	0.013	0.020	1.33	2.12	7.3	0.015	0.023	1.74	2.37
54	18	3	9	0.020	0.200	0.52	4.33	10	0.000	0.000	0.04	0.07	10	0.000	0.000	0.08	0.13
72	18	4	5	0.050	0.100	3.43	7.36	0	0.060	0.120	7.71	8.96	0	0.098	0.130	8.02	9.66
90	18	5	10	0.000	0.000	0.22	0.73	4	0.014	0.040	8.31	16.49	0	0.027	0.040	12.76	15.75
180	18	10	10	0.000	0.000	0.05	0.21	10	0.000	0.000	0.23	1.90	9	0.001	0.010	2.33	20.78
270	18	15	10	0.000	0.000	0.03	0.06	10	0.000	0.000	0.06	0.15	10	0.000	0.000	0.10	0.26
360	18	20	10	0.000	0.000	0.03	0.03	10	0.000	0.000	0.06	0.10	10	0.000	0.000	0.10	0.27
average			9.0	0.012	0.050	0.71	2.12	7.3	0.012	0.027	2.74	4.61	6.5	0.021	0.030	3.90	7.81
60	20	3	6	0.090	0.300	2.66	5.54	9	0.024	0.240	0.60	5.32	10	0.000	0.000	0.11	0.22
80	20	4	7	0.030	0.100	3.49	10.82	0	0.068	0.100	10.55	12.38	0	0.065	0.100	11.18	14.28
100	20	5	10	0.000	0.000	3.76	33.20	5	0.010	0.020	12.38	26.07	1	0.027	0.050	19.96	34.83
200	20	10	10	0.000	0.000	0.02	0.05	10	0.000	0.000	0.06	0.14	10	0.000	0.000	0.21	0.49
300	20	15	10	0.000	0.000	0.04	0.12	10	0.000	0.000	0.07	0.14	10	0.000	0.000	0.10	0.15
400	20	20	10	0.000	0.000	0.06	0.25	10	0.000	0.000	0.06	0.16	10	0.000	0.000	0.12	0.24
average			8.8	0.020	0.067	1.67	8.33	7.3	0.017	0.060	3.95	7.37	6.8	0.015	0.025	5.28	8.37
75	25	3	2	0.160	0.200	7.62	9.39	10	0.000	0.000	0.09	0.17	10	0.000	0.000	0.15	0.23
100	25	4	8	0.020	0.100	7.15	18.96	0	0.060	0.100	19.08	23.51	0	0.064	0.100	19.97	24.82
125	25	5	10	0.000	0.000	1.38	2.93	10	0.000	0.000	7.97	26.50	4	0.008	0.020	36.75	72.70
250	25	10	10	0.000	0.000	0.15	0.47	10	0.000	0.000	2.68	24.79	10	0.000	0.000	0.31	0.67
375	25	15	10	0.000	0.000	0.16	0.72	10	0.000	0.000	0.09	0.13	10	0.000	0.000	0.19	0.59
average			8.0	0.036	0.060	3.29	6.49	8.0	0.012	0.020	5.98	15.02	6.8	0.014	0.024	11.47	19.80
90	30	3	0	0.180	0.200	13.65	14.48	1	0.148	0.200	13.43	15.89	6	0.059	0.190	5.86	15.84
120	30	4	4	0.060	0.100	23.07	42.53	0	0.054	0.100	37.67	44.47	0	0.069	0.090	39.86	48.83
150	30	5	10	0.000	0.000	1.99	6.65	10	0.000	0.000	11.95	42.28	2	0.011	0.020	78.27	152.07
300	30	10	10	0.000	0.000	0.09	0.37	10	0.000	0.000	0.78	3.83	10	0.000	0.000	1.37	8.73
average			6.0	0.060	0.075	9.70	16.01	5.3	0.050	0.075	15.95	26.62	4.5	0.035	0.075	31.34	56.37
overall average			8.7	0.018	0.041	1.57	3.56	7.6	0.015	0.030	3.15	5.81	6.9	0.015	0.027	5.48	9.60

Table 2.7: Results for real world PCB instances

							Percentage error						
	n	m	k	L	z	opt	LPT_k	MS_k	$MS2_k$	*Scatter 0*	*Scatter 1*	*B&B*	time
PCB_1	403	5	81	115340	115341	0	0.0139	16.3092	0.1621	0.0009	0.0009	0.0009	5.46
	403	10	41	57670	57672	0	0.1075	15.5419	9.7312	0.0035	0.0035	0.0035	10.36
	403	20	21	32274	32274	1	0.0000	10.7920	10.8106	0.0000	0.0000	0.0000	0.01
	1341	5	269	512403	512403	1	0.0135	13.0760	12.4379	0.0000	0.0000	0.0000	1.13
	1341	10	135	256202	256203	0	0.0297	12.9757	6.2041	0.0004	0.0004	0.0004	67.55
	1341	20	68	128101	128103	0	0.0390	12.3434	10.7954	0.0016	0.0016	0.0016	130.62
	1417	5	284	467925	467925	1	0.0077	14.4190	13.8144	0.0000	0.0000	0.0000	0.64
	1417	10	142	233963	233964	0	0.0043	14.3809	11.1663	0.0004	0.0004	0.0004	69.62
	1417	20	71	116982	116982	1	0.1359	14.4868	14.4407	0.0000	0.0000	0.0000	2.79
	1312	5	263	446266	446268	0	0.0139	14.0739	1.6817	0.0004	0.0004	0.0004	34.95
	1312	10	132	223133	223134	0	0.0341	13.1778	4.0191	0.0004	0.0004	0.0004	63.23
	1312	20	66	111567	111567	1	0.0215	14.4855	14.4371	0.0000	0.0000	0.0000	0.49
	average						0.0351	13.8385	9.1417	0.0006	0.0006	0.0006	32.24
PCB_2	314	5	63	114324	114324	1	0.0184	12.4541	11.3546	0.0000	0.0000	0.0000	0.05
	314	10	32	57162	57162	1	0.0052	12.0710	11.4884	0.0000	0.0000	0.0000	0.01
	314	20	16	31965	31965	1	0.0000	8.9911	8.6532	0.0000	0.0000	0.0000	0.01
	972	5	195	512007	512007	1	0.0129	11.8006	11.1432	0.0000	0.0000	0.0000	1.36
	972	10	98	256004	256005	0	0.0238	10.8190	8.9522	0.0004	0.0004	0.0004	45.17
	972	20	49	128002	128004	0	0.0156	11.0287	9.0881	0.0016	0.0016	0.0016	84.79
	1084	5	217	459495	459495	1	0.0104	13.1198	12.5747	0.0000	0.0000	0.0000	1.00
	1084	10	109	229749	229749	1	0.0261	13.0264	10.7047	0.0000	0.0000	0.0000	6.23
	1084	20	55	114876	114876	1	0.0470	12.6893	10.8378	0.0000	0.0000	0.0000	9.35
	1056	5	212	445266	445266	1	0.0168	13.0520	12.4375	0.0000	0.0000	0.0000	1.44
	1056	10	106	222633	222633	1	0.0364	13.1625	13.0479	0.0000	0.0000	0.0000	1.55
	1056	20	53	111317	111318	0	0.0252	13.0825	11.3253	0.0009	0.0009	0.0009	109.87
	average						0.0198	12.1081	10.9673	0.0002	0.0002	0.0002	21.74
	overall average						0.0275	12.9733	10.0545	0.0004	0.0004	0.0004	26.99

Chapter 3

On the k_i-Partitioning Problem

[1]

We consider the problem of partitioning a set of positive integers values into a given number of subsets, each having an associated cardinality limit, so that the maximum sum of values in a subset is minimized, and the number of values in each subset does not exceed the corresponding limit. The problem is related to scheduling and bin packing problems. We give combinatorial lower bounds, reduction criteria, constructive heuristics, a scatter search approach, and a lower bound based on column generation. The outcome of extensive computational experiments is presented.

Key words: partitioning, cardinality constraints, scheduling, parallel machines, bin packing, scatter search, column generation.

3.1 Introduction

Given n *items* I_j $(j = 1, \ldots, n)$, each characterized by an integer non negative *weight* w_j, and m positive integers k_i $(i = 1, \ldots, m)$ with $m < n \leq \sum_{i=1}^{m} k_i$, the *$k_i$-Partitioning Problem* ($k_i$-PP) is to partition the items into m *subsets* $S_1, \ldots, S_m$ so that $|S_i| \leq k_i$ $(i = 1, \ldots, m)$ and the maximum total weight of a subset is a minimum. The problem was introduced by Babel, Kellerer and Kotov [5] and finds possible applications, e.g., in Flexible Manufacturing Systems. Assume that we have to execute a set of operations of n different types, and that the operations of type j, requiring in total a time w_j, must be assigned to the same cell: If the capacity of the specific tool magazine of each cell imposes a limit on the number of types of operation the cell can perform, then *k_i-PP* models the problem of completing the process in minimum total time.

A famous scheduling problem (usually denoted as $P||C_{\max}$) asks for assigning n *jobs*, each having an integer non negative *processing time* w_j, to m identical parallel *machines* M_i $(i = 1, \ldots, m)$, each of which can process at most one job at a time, so as to minimize their total completion time (*makespan*). By associating items to jobs and subsets to machines, it is clear that *k_i-PP* is the generalization of $P||C_{\max}$ arising when an additional constraint imposes an upper bound k_i on the number of jobs that can be processed by machine M_i. Since $P||C_{\max}$ is known to be strongly NP-hard, the same holds for k_i-PP.

Another special case of k_i-PP, that also generalizes $P||C_{\max}$, is the $P|\# \leq k|C_{\max}$ scheduling problem, in which an identical limit k is imposed on the maximum number of jobs that can be assigned to any machine. Upper and lower bounds for this problem have been developed by Babel, Kellerer and Kotov [5], Dell'Amico and Martello [38] and Dell'Amico, Iori and Martello [39].

[1] The results of this chapter appears in: M. Dell'Amico, M. Iori, M. Monaci and S. Martello, Lower Bounds and Heuristic Algorithms for the k_i- Partitioning Problem, *European Journal of Operational Research, Special Issue on Stochastic and Heuristic Methods in Optimization*, 2004 (to appear), [40].

The *Bin Packing Problem* (*BPP*) too is related to k_i-PP. Here we are given n items, each having an associated integer non negative *weight* w_j, and an unlimited number of identical containers (*bins*) of *capacity* c: the problem is to assign all items to the minimum number of bins so that the total weight in each bin does not exceed the capacity. Problem *BPP* can be seen as a 'dual' of $P||C_{\max}$: By determining the minimum c value for which an m-bin *BPP* solution exists, we also solve the corresponding $P||C_{\max}$ problem. By introducing a limit k on the number of items that can be assigned to any bin, we similarly obtain a dual of $P|\#\leq k|C_{\max}$. In order to obtain a dual of k_i-PP, we can impose the given limits k_i $(i=1,\ldots,m)$ to the first m bins, and a limit equal to one to all other bins.

The dual relations above have been used to obtain heuristic algorithms and lower bounds for $P||C_{\max}$ (Coffman, Garey and Johnson [28], Hochbaum and Shmoys [90], Dell'Amico and Martello [41]) and $P|\#\leq k|C_{\max}$ (Dell'Amico and Martello [38]).

In this paper we study upper and lower bounds for k_i-PP, either obtained by generalizing algorithms from the literature so as to handle the cardinality constraints, or originally developed for the considered problem. In Section 3.2 we present lower bounds and reduction criteria. In Section 3.3 we examine generalizations of heuristic algorithms and of a scatter search approach. In Section 3.4 we propose a lower bound based on a column generation approach, that makes use of the above mentioned relations with *BPP*. The effectiveness of the proposed approaches is computationally analyzed in Section 3.5 through extensive computational experiments on randomly generated data sets.

Without loss of generality, we will assume in the following that items I_j are sorted by non-increasing w_j value, and subsets S_i by non-decreasing k_i value.

3.2 Lower bounds and reduction criteria

By introducing binary variables x_{ij} $(i=1,\ldots,m,\ j=1,\ldots,n)$ taking the value 1 iff item I_j is assigned to subset S_i, an ILP model of k_i-*PP* can be written as

$$\min z \tag{3.1}$$

$$\sum_{j=1}^{n} w_j x_{ij} \leq z \quad (i=1,\ldots,m) \tag{3.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (j=1,\ldots,n) \tag{3.3}$$

$$\sum_{j=1}^{n} x_{ij} \leq k_i \quad (i=1,\ldots,m) \tag{3.4}$$

$$x_{ij} \in \{0,1\} \quad (i=1,\ldots,m; j=1,\ldots,n) \tag{3.5}$$

where variable z represents the makespan.

In the following we will denote by

$$free = \sum_{i=1}^{m} k_i - n \tag{3.6}$$

the total number of unused feasible assignments to the subsets (with respect to the cardinality constraints) in any solution. Given a complete or partial solution x to (3.1)-(3.5),

$$W_i = \sum_{j=1}^{n} w_j x_{ij} \quad (i=1,\ldots,m) \tag{3.7}$$

$$card_i = \sum_{j=1}^{n} x_{ij} \quad (i=1,\ldots,m) \tag{3.8}$$

will denote, respectively, the total weight of the items currently assigned to subset S_i, and their number. In the next section we present lower bounds and reduction algorithms based on the combinatorial structure of the problem. In our approach these computations, together with those of the heuristics of Section 3.3.1, are performed at the beginning of the scatter search approach described in Section 3.3.2. If an optimal solution is not obtained, the bound is improved through a computationally heavier approach, based on column generation, described later in Section 3.4.

3.2.1 Combinatorial lower bounds

Since $P|\#\leq k|C_{\max}$ is a special case of k_i-PP, any lower bound for the former problem with k set to k_m (the largest cardinality limit) is also valid for the latter. We will denote by

$$L_{|\#\leq k_m|} = \max(L_3^k, L_{BKK}, L_{HS}) \tag{3.9}$$

the best among three bounds from the literature, used in Dell'Amico, Iori and Martello [39] for $P|\#\leq k|C_{\max}$. These bounds will not be described here for the sake of conciseness: The complete description can be found in [39] and in Dell'Amico and Martello [38], Babel, Kellerer and Kotov [5] and Hochbaum and Schmoys [90].

The following lower bounds explicitly take into account cardinality constraints (3.4).

Theorem 3.1. *Given any instance of k_i-PP, the value*

$$L_1 = w_1 + \sum_{j=n-k_1+free+2}^{n} w_j \tag{3.10}$$

is a valid lower bound on the optimal solution value.

Proof. Assume by simplicity that $k_1 - free > 1$, and consider the item with maximum weight w_1. By (3.6), in any feasible solution such item will be assigned to a subset containing no less than $k_1 - free - 1$ other items. The thesis follows, since in (3.10) it is assumed that the other items in such subset are the smallest ones. (If $k_1 - free \leq 1$, note that a summation $\sum_{j=\alpha}^{\beta} w_j$ is considered to produce the value zero if $\beta < \alpha$.) □

Theorem 3.2. *Given any instance of k_i-PP, the value*

$$L_2 = \max_{\ell=1,\ldots,m} \left\{ \left\lceil \sum_{j=n-k(\ell)+free+1}^{n} w_j/\ell \right\rceil \right\}, \tag{3.11}$$

where $k(\ell) = \sum_{i=m-\ell+1}^{m} k_i$, is a valid lower bound on the optimal solution value.

Proof. Let ℓ be any integer between 1 and m, and consider the last ℓ subsets. Assume by simplicity that $free$ is strictly smaller than $k(\ell)$, the sum of the cardinality limits for such subsets. This implies that, even if the other subsets contain the maximum possible number of items, the last ℓ subsets will contain, in total, no less than $k(\ell) - free$ items. In the best case: (i) these items will be the smallest ones, and (ii) they will be partitioned, among the last ℓ subsets, so as to produce identical total weights. Hence $\lceil \sum_{j=n-k(\ell)+free+1}^{n} w_j/\ell \rceil$ is valid lower bound for any ℓ value, and the validity of (3.11) follows. □

If the summation in (3.10) has zero value, L_1 gives the obvious $P||C_{\max}$ bound w_1. Another immediate lower bound that comes from $P||C_{\max}$ (hence, does not take into account the cardinality constraints) but is not dominated by L_1 nor by L_2 is

$$L_3 = \max\left(w_m + w_{m+1}, \left\lceil \sum_{j=1}^{n} w_j/m \right\rceil \right) \tag{3.12}$$

Our overall combinatorial bound is thus

$$L_C = \max(L_{|\#\leq k_m|}, L_1, L_2, L_3) \tag{3.13}$$

3.2.2 Reduction criteria

Remind that the items are sorted by non-increasing w_j value, and the subsets by non-decreasing k_i value, and observe that, if $k_1 = 1$, there exists an optimal solution in which $S_1 = \{I_1\}$. This reduction process can be iterated, as shown in Figure 3.1.

Procedure Reduction1
1. $i := 1$;
2. **while** $k_i = 1$ **do** $S_i = \{I_i\}$, $i := i + 1$;
3. define a reduced instance by removing the first $i - 1$ items and subsets
end

Figure 3.1: Reduction1

Reduction1 can be performed before any other computation. Once a lower bound L (e.g., L_C, see (3.13)) has been computed, a more powerful reduction can be obtained, based on the following considerations. First observe that if the total weight of the largest k_1 items does not exceed L, then there exists an optimal solution in which $S_1 = \{I_1, \ldots, I_{k_1}\}$. This consideration can be extended to the first $\widetilde{m}$, say, subsets and the first $\sum_{i=1}^{\widetilde{m}} k_i$ items: If we can obtain a feasible partial solution of value not exceeding L, then these items can be assigned to these subsets as in the solution found, and both subsets and items can be removed from the instance. Again, the process can be iterated, as shown in Figure 3.2.

Procedure Reduction2(L)
1. $firstS := lastS := 1$, $firstI := 1$, $lastI = k_1$;
2. **repeat**
   2.1 let $\widetilde{x}$ be a feasible solution of value $\widetilde{z}$ to the sub-instance induced by subsets $\{S_{firstS}, \ldots, S_{lastS}\}$ and items $\{I_{firstI}, \ldots, I_{lastI}\}$;
   2.2 **if** $\widetilde{z} \leq L$ **then**
   assign items $\{I_{firstI}, \ldots, I_{lastI}\}$ to subsets $\{S_{firstS}, \ldots, S_{lastS}\}$ as in $\widetilde{x}$;
   $firstS := lastS + 1$, $lastS := firstS$;
   $firstI := lastI + 1$, $lastI := firstI + k_{firstS} - 1$;
   else $lastS := lastS + 1$, $lastI := lastI + k_{lastS}$
   endif
   until stopping conditions are met;
3. define a reduced instance by removing the first $lastI$ items and $lastS$ subsets
end

Figure 3.2: Reduction2

In our implementation the stopping condition is ($\widetilde{z} > L$ **and** $lastI \geq n/2$). When the sub-instance includes just one subset, it can be trivially solved. The solution of sub-instances with more than one subset is obtained through the heuristics of the next section. Whenever Reduction2 manages to reduce the current instance, the lower bound is re-computed: If it increases, the procedure is re-executed.

3.3 Heuristic algorithms

Problem $P||C_{\max}$ has been attacked with several constructive approximation algorithms and with some metaheuristic approaches (see, e.g., the surveys by Lawler et al. [104], Hoogeveen, Lenstra and van

de Velde [92], Mokotoff [125]). The constructive heuristics can be subdivided into the following three main classes.

- *List Scheduler*: After sorting the jobs according to a given rule, the algorithm considers one job at a time and assigns it to the machine M_i with minimum current load, without introducing idle times. For $P||C_{\max}$ one of the more effective sorting rules is the so called *Longest Processing Time* (*LPT*) introduced by Graham [77], that orders the jobs by non-increasing processing times. The probabilistic analysis in Coffman, Lueker and Rinnooy Kan [29] shows that, under certain conditions, the solution produced by *LPT* is asymptotically optimal.
- *Dual algorithms*: These methods exploit the 'duality' with *BPP* outlined in Section 3.1, using two possible strategies. Both strategies start with a tentative solution value $\widetilde{c}$ and solve the corresponding *BPP* instance with bin capacity $\widetilde{c}$, by means of some heuristic. If the solution uses more that m bins then: (i) the first strategy re-assigns to bins $1, \ldots, m$ the items assigned to bins $m+1, m+2, \ldots$ using some greedy method; (ii) the second strategy finds, through binary search, the smallest $\widetilde{c}$ value for which the induced *BPP* instance uses no more than m bins. An example of dual approach implementing the first strategy is the *Multi Subset* (*MS*) method by Dell'Amico and Martello [41], while methods using the second strategy are the *Multi Fit* algorithm (*MF*) proposed by Coffman, Garey and Johnson [28], and the ϵ-dual method by Hochbaum and Shmoys [90].
- *Mixed algorithms*: The main idea of these methods is to combine two or more solution techniques, by switching from one to another during the construction of the solution. The rationale behind these methods is to try and catch the best of each basic algorithm. For example, it is well known that *LPT* is able to construct partial solutions with values of the machine loads very close to each other, but it fails in this attempt when assigning the last jobs. A mixed algorithm tries to overcome this problem by using *LPT* for assigning the first, say, $\hat{n} < n$ jobs, then completes the solution using another rule (see e.g. Mokotoff, Jimeno and Gutiérrez [100]).

The solution obtained with one of the methods above can be improved through re-optimization and local search. It is quite surprising that several metaheuristic approaches can be found in the literature for variants of $P||C_{\max}$ arisng, e.g., when there are sequence depending setups, or the objective function is the sum of the completion times of the last job of each machine, but very few algorithms have been proposed for the pure $P||C_{\max}$ problem. Finn and Horowitz [55] introduced a polynomial improvement algorithm called *0/1 interchange*. Hübscher and Glover [96] proposed a tabu search algorithm, Fatemi-Ghomi and Jolai-Ghazvini [50] a local search approach based on 2-exchanges. Mendes et al. [123] compared a tabu search approach with a memetic algorithm, while Frangioni, Necciari and Scutellà [56] presented a local search algorithm based on multiple exchanges within large neighborhoods.

For $P|\#\leq k|C_{\max}$, the only metaheuristic in the literature is, to our knowledge, the scatter search algorithm proposed by Dell'Amico, Iori and Martello [39].

In the next sections we describe constructive heuristics and a scatter search algorithm for k_i-PP, that were obtained by generalizing constructive heuristics for $P||C_{max}$ and the scatter search algorithm for $P|\#\leq k|C_{\max}$. For the constructive heuristics the modifications are aimed to handle the cardinality constraints, while for the scatter search they consist in generalizing the simpler cardinality contraints of $P|\#\leq k|C_{\max}$ to our case. We give in the following a synthetical description of the resulting k_i-*PP* algorithms.

3.3.1 Constructive Heuristics

We obtained heuristics for k_i-*PP* by generalizing methods for $P||C_{max}$. The following algorithms were obtained (see (3.7) and (3.8) for the definitions of W_i and $card_i$).

LPT-k_i: This is an implementation of the *LPT* list scheduler, with an additional condition imposing that no more than k_i items are assigned to subset S_i. The items are initially sorted by non-increasing

w_j values. At each iteration, the next item is assigned to the subset S_i with minimum total weight W_i among those satisfying $card_i < k_i$ (breaking ties by lower k_i value).

GH-k_i: This is an iterative mixed approach derived from the *Gap Heuristics* by Mokotoff, Jimeno and Gutiérrez [100] for $P||C_{max}$. At each iteration, a procedure that builds up a complete solution is executed with different values of a parameter λ. This procedure starts by assigning the items to the m subsets with the *LPT-k_i* method but, as soon as the minimum weight of a subset reaches λ, it switches to a different rule: Assign the current item to the subset S_i, among those with $card_i < k_i$, for which the resulting total weight W_i is as close as possible to lower bound L_C. The best solution obtained is finally selected.

MS1-k_i: The method is obtained from the dual algorithm *MS* by Dell'Amico and Martello [41]. In the first phase, this algorithm approximately solves the associated *BPP* instance by filling one bin at a time through the solution of an associated *Subset Sum Problem* (*SSP*): Given n integers and a single bin of capacity c, find the subset of integers whose sum is closest to, without exceeding, c. To adapt *MS* to *k_i-PP* it is then enough to modify the procedure used to define each subset S_i (bin) so that it only produces solutions satisfying $|S_i| \leq k_i$. The procedure used in our implementation was approximation algorithm G^2 by Martello and Toth [117], that builds up an *SSP* solution by selecting one item at a time: It is then easy to modify it so as to handle the additional constraint. In the second phase, *MS* uses a greedy method to re-assign the items (if any) not inserted in the first m bins, and again the cardinality constraint is easily embedded.

MS2-k_i: It differs from *MS1-k_i* only in the method used to solve the *SSP* instances, that is here the simpler and faster algorithm G^1 in [117].

MS3-k_i: This is a hybrid branch-and-bound/*MS* method. We start with a branch-decision tree truncated at level ℓ. At each level $l \leq \ell$ we assign item I_l, in turn, to all already initialized subsets, and (possibly) to a new one, provided the corresponding cardinality constraints are not violated: Each node has then an associated partial solution consisting of the first l items. We consider all the partial solutions of level ℓ: Each of them is completed by applying *MS1-k_i* and *MS2-k_i* to the remaining $n - \ell$ unassigned items, and the best solution is finally selected. In our implementation we set $\ell = 5$.

MS-k_i: Execute *MS1-k_i*, *MS2-k_i* and *MS3-k_i*, and select the best result.

3.3.2 Scatter Search

Scatter Search is a metaheuristic technique whose founding ideas go back to the seminal works of Glover [62, 63] in the early Sixties. Apparently these ideas were neglected for more than 20 years and resumed in Glover [64], where Scatter Search was presented for the first time. In the Eighties and the Nineties the method was successfully applied to solve a large set of problems. The basic idea (see Glover [69]) can be outlined in the following three steps:

1. generate a starting set of solutions (the *reference set*), possibly using problem dependent heuristics;
2. create new solutions through combinations of subsets of the current reference solutions;
3. extract a collection of the best solutions created in Step 2 and iterate on Step 2, unless a stopping criterion holds.

This basic procedure has been improved in several ways. First of all, the reference set is usually divided into two subsets: The first one containing solutions with a high "quality", the second one containing solutions very "diverse" from the others. A second refinement consists in applying some re-optimization algorithm to each solution before deciding if it has to be stored in the reference set. Other improvements concern the methods used to generate the starting solutions, to combine the

solutions and to update the reference set. We based our implementation on a classical template (see Laguna [101], Glover, Laguna and Martí [73]) that was already used in Dell'Amico, Iori and Martello [39] for $P|\#\leq k|C_{\max}$, summarized in Figure 3.3.

Procedure Scatter
1. Randomly generate a set T of solutions and improve them through *intensification*.
2. Compute the *fitness* of each solution, i.e., a measure of its "quality".
3. Create a reference set R of r distinct solutions by selecting from T the q solutions with highest fitness, and the d solutions with highest *diversity*.
4. Evolve the reference set R through the following steps:
   4.1 *Subset generation*: generate a family $\mathcal{G}$ of subsets of R.
   4.2 **while** $\mathcal{G}\neq\emptyset$ **do**
   extract a subset from $\mathcal{G}$ and obtain a solution s through *combination*;
   improve s through *intensification*;
   execute the *reference set update* on R
   endwhile;
   4.3 **if** *stopping criteria* are not met **then go to** 4.1;

end

Figure 3.3: Scatter Search

On the basis of the outcome of extensive computational experiments, we set the size of the initial set T to 100, while the reference set R has size $r=q+d=18$ (with $q=10$ and $d=8$). In the reference set we maintain separated the high-quality solutions (first q solutions) and the high-diversity solutions (last d solutions). Solutions $1,\ldots,q$ are ordered by decreasing quality, while solutions $q+1,\ldots,r$ are ordered by increasing diversity (i.e., the best solution is the first one and the most diverse is the last one).

In the following we give some details on the implementation of *fitness* calculation, *diversity* evaluation, *intensification*, *subset generation*, *combination* method, *reference set update* and *stopping criteria*.

Fitness. Let $z(s)$ be the value of a solution s. The corresponding fitness is defined as $f(s)=z(s)/(z(s)-L)$, where L denotes the best lower bound value obtained so far. We have chosen this function instead of the pure solution value in order to obtain a less flat search space in which differences are highlighted, so the search can be directed towards more promising areas.

Diversity. Given two solutions, s and t, and an item, I_j, we compute

$$\delta_j(s,t)=\begin{cases}1 & \text{if } I_j \text{ is assigned to the same subset in solutions } s \text{ and } t;\\ 0 & \text{otherwise}\end{cases} \tag{3.14}$$

and establish the diversity of s from the solutions in R as the minimum 'distance' of s from a solution of the set, defined as

$$d(s)=\min_{t\in R}\left\{\sum_{j=1}^{2m}\delta_j(s,t)\right\} \tag{3.15}$$

Intensification. Given a solution s, we apply a sequence of re-optimization procedures obtained by generalizing those introduced in Dell'Amico, Iori and Martello [39] for $P|\#\leq k|C_{\max}$.

Procedure *MOVE* considers one subset S_i at a time and tries to move large items from S_i to other subsets. It starts with the largest item, say I_j, currently assigned to S_i and looks for the first subset S_h $(h\neq i)$ such that $card_h<k_h$ and $W_h+w_j<W_i$. If such S_h exists, item I_j is moved to S_h and the procedure is re-started; otherwise the next largest item of S_i is selected and the search is iterated.

Procedure *EXCHANGE* works as *MOVE*, with just one difference: The selected large item $I_j \in S_i$ is not just moved to S_h but exchanged with an item $I_g \in S_h$, provided that $w_g < w_j$ and $W_h - w_g + w_j < W_i$.

In the special case $n = \sum_{i=1}^{m} k_i$, two additional re-optimization procedures, *MIX1* and *MIX2*, are applied. *MIX1* builds a new partial solution by initially assigning the first $\overline{n}$ ($\overline{n} < n$) items as in the given solution, and then considering the subsets according to non-increasing W_i values. For each S_i, the quantity $gap_i = k_i - card_i$ (number of items that can be still assigned to S_i) is evaluated: if $gap_i > \overline{k}$ (for a given parameter $\overline{k}$), then the $(gap_i - \overline{k})$ smallest items are assigned to S_i. In any case, S_i is completed with the addition of $\min\{gap_i, \overline{k}\}$ items that are selected, through complete enumeration, in such a way that the resulting W_i value is as close as possible to L. Procedure *MIX2* differs from *MIX1* only in the construction of the initial partial solution: Given a prefixed parameter $\widetilde{k}$, each subset S_i is initialized with the first (largest) $\min\{\widetilde{k}, k_i\}$ items as in the original solution. The values of the parameters were experimentally determined as: $\overline{k} = 3$, $\overline{n} = \max(m, n-3m)$ and $\widetilde{k} = \max\{0, (k_m - 3)\}$.

Subset generation. We adopted the classical multiple solution method (see, e.g., Glover, Laguna and Martí [73]), that generates, in sequence, all 2-element subsets, the 3-element subsets that are obtained by augmenting each 2-element subset to include the best solution not already belonging to it, the 4-element subsets that are obtained by augmenting each 3-element subset to include the best solution not already belonging to it and the i-element subsets (for $i = 5, \ldots, r$) consisting of the best i elements.

Combination. Given a number of distinct solutions $s_1, s_2, \ldots$, we define an $m \times n$ fitness matrix $\mathcal{F}$ with $\mathcal{F}_{ij} = \sum_{a \in A_{ij}} f(s_a)$, where A_{ij} is the index set of the solutions where item I_j is assigned to subset S_i. We first construct γ solutions (γ being a given parameter) through a random process that, for $j^* = 1, \ldots, n$, assigns item I_{j^*} to subset S_{i^*} with probability $\mathcal{F}(i^*, j^*) / \sum_{i=1}^{m} \mathcal{F}(i, j^*)$. If the resulting subset S_{i^*} has k_{i^*} items assigned, then we set $\mathcal{F}(i^*, j) = 0$ for $j = 1, \ldots, n$ (so S_i^* is not selected at the next iterations). If for the current item I_j^* we have $\sum_{i=1}^{m} \mathcal{F}(i, j^*) = 0$, then the item is assigned to the subset with minimum weight W_i among those that have not reached the cardinality limit. We finally select the best-quality solution among the γ solutions generated. In our implementation we set $\gamma = 3$ for $n < 100$ and $\gamma = 1$ for $n \geq 100$.

Reference set update. The *dynamic reference set update* (see, e.g., Glover, Laguna and Martí [73]) has been introduced to update the reference set by maintaining a good level of quality and diversity. A solution enters R if its fitness is better than that of the q-th best solution (the worst of the high-quality solutions maintained in R), or its diversity (computed through (3.14) and (3.15)) is higher than that of the $(q+1)$-th solution (the less diverse solution of R).

Stopping criteria. We terminate the search if: (i) the current best solution has value equal to lower bound L; or (ii) no reference set update occurs at Step 4.; or (iii) Step 4. has been executed b times, with $b = (10, 5, 1)$, for $n < 100$, $100 \leq n < 400$ and $n > 400$, respectively.

3.4 Column generation lower bound

In this section we introduce a lower bound obtained by iteratively solving, through column generation, the LP relaxation of the following variant of multiple subset-sum problem.

SSPK(c): Given the input of an instance of k_i-*PP* and a threshold value c, assign items to the subsets so that no subset has a total weight exceeding c or a total number of items exceeding its cardinality limit, and the number of unassigned items is a minimum.

Let U be the value of the best heuristic solution produced by the algorithms of Section 3.3, and L the best lower bound value obtained so far. A possible approach for solving k_i-*PP* could attempt a 'dual'

strategy consisting of a specialized binary search between L and U: at each iteration one considers a threshold value $c = \lfloor (L+U)/2 \rfloor$, and solves *SSPK*($c$). If the optimal solution has value zero, a solution of value c to k_i-*PP* has been found, so it is possible to set $U = c$ and iterate with a new (smaller) value of c. If instead the optimal solution value is greater than zero, we know that no feasible solution of value c exists for k_i-PP, so it is possible to set $L = c + 1$ and iterate with a new (higher) value of c. The search stops with an optimal solution when $L = U$.

In the next section we give an ILP model for *SSPK*(c), derived from the set covering formulation of *BPP*. The model will be used in Section 3.4.2 to obtain a lower bound for k_i-*PP* through column generation.

3.4.1 An ILP model for SSPK(c)

Let $K = \{\kappa_1, \ldots, \kappa_{\overline{m}}\}$ be the set of distinct k_i values (sorted by increasing κ_i values), and assume that $\kappa_0 = 0$. For each value $\kappa_r \in K$ let

$$\mathcal{G}^r = \{S_i : k_i \geq \kappa_r\} \tag{3.16}$$

be the family of those subsets that can contain κ_r or more items, and

$$\mathcal{P}^r(c) = \{P \subseteq \{I_1, \ldots, I_n\} : \kappa_{r-1} < |P| \leq \kappa_r \text{ and } \sum_{I_j \in P} w_j \leq c\} \tag{3.17}$$

be a family of item sets (*patterns*) that can be feasibly assigned to a member of $\mathcal{G}^r$ but not to a member of $\mathcal{G}^{r-1} \setminus \mathcal{G}^r$. Observe that, by (3.17), $\{\mathcal{P}^1(c), \ldots, \mathcal{P}^{\overline{m}}(c)\}$ is a partition of all patterns that have total weight not greater than c. For any $\kappa_r \in K$ and $j \in \{1, \ldots, n\}$, let $\mathcal{P}^r_j(c) \subseteq \mathcal{P}^r(c)$ be the set of those patterns of $\mathcal{P}^r(c)$ that contain item I_j.

Let us introduce binary variables

$$x_P = \begin{cases} 1 & \text{if pattern } P \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \qquad (P \in \mathcal{P}^r(c), \kappa_r \in K) \tag{3.18}$$

and

$$y_j = \begin{cases} 1 & \text{if item } I_j \text{ is not assigned to any subset} \\ 0 & \text{otherwise} \end{cases} \qquad (j = 1, \ldots, n) \tag{3.19}$$

We obtain the ILP model

$$SSPK(c) \qquad \min \sum_{j=1}^{n} y_j \tag{3.20}$$

$$\sum_{\kappa_r \in K} \sum_{P \in \mathcal{P}^r_j(c)} x_P + y_j = 1 \quad (j = 1, \ldots, n) \tag{3.21}$$

$$\sum_{i \geq r} \sum_{P \in \mathcal{P}^i(c)} x_P \leq |\mathcal{G}^r| \quad (\kappa_r \in K) \tag{3.22}$$

$$y_j \in \{0, 1\} \quad (j = 1, \ldots, n) \tag{3.23}$$

$$x_P \in \{0, 1\} \quad (P \in \mathcal{P}^r(c), \kappa_r \in K) \tag{3.24}$$

Constraints (3.21) impose that each item I_j is assigned to exactly one subset, or not assigned at all. Constraints (3.22) impose, for each $\kappa_r \in K$, the selection of at most $|\mathcal{G}^r|$ patterns (see (3.16)) among those containing κ_r items or more. Objective function (3.20) minimizes the number of unassigned items.

As we are just interested in computing a lower bound L_{CG} on the optimal k_i-*PP* solution, instead of optimally solving each *SSPK*(c) instance we solve its continuous relaxation through the column

generation approach described in the next section. In this case, the binary search outlined above has to be modified as follows. If the optimal LP solution to *SSPK*(c) has value greater than zero, it is still possible to set $L = c + 1$ and iterate with the higher value of c. If instead the optimal LP solution to *SSPK*(c) has value zero, two possibilities arise: (i) if the solution is integer (i.e., also optimal for *SSPK*(c)), it is still possible to set $U = c$ and iterate with the smaller value of c; (ii) otherwise no further improvement is possible, so the search stops with the current L value as L_{CG}. Further observe that in case (i) the incumbent solution value can be possibly improved.

3.4.2 Column Generation

In this section we discuss methods for handling the LP relaxation of *SSPK*(c) when the number of patterns is too large to explicitly include all the corresponding variables into the model. The LP relaxation of *SSPK*(c) (*master problem*) is given by (3.20)-(3.22) and

$$y_j \geq 0 \quad (j = 1, \ldots, n) \tag{3.25}$$

$$x_P \geq 0 \quad (P \in \mathcal{P}^r(c), \kappa_r \in K) \tag{3.26}$$

(Note that constraints $y_j \leq 1$ and $x_P \leq 1$ would be redundant.) By associating dual variables π_j and σ_r to constraints (3.21) and (3.22), respectively, we obtain the dual:

$$\max \sum_{j=1}^{n} \pi_j - \sum_{\kappa_r \in K} |\mathcal{G}^r|\, \sigma_r \tag{3.27}$$

$$\sum_{j \in P} \pi_j - \sum_{i \leq r} \sigma_i \leq 0 \quad (P \in \mathcal{P}^r(c), \kappa_r \in K) \tag{3.28}$$

$$\pi_j \leq 1 \quad (j = 1, \ldots, n) \tag{3.29}$$

$$\sigma_r \geq 0 \quad (\kappa_r \in K) \tag{3.30}$$

A column generation approach starts by solving a restriction of the LP relaxation of *SSPK*(c), obtained by only considering a small subset of variables x_P (*restricted master*). The iterative phase consists in checking if the solution (π^*, σ^*) of the dual of the current restricted master satisfies all constraints (3.28): If this is the case, the current solution value provides a valid lower bound for *SSPK*(c). Otherwise primal variables corresponding to a subset of violated constraints (3.28) are added to the restricted master (*column generation*), and the process is iterated.

The check is performed by looking for primal variables (if any) with negative reduced cost. If each item I_j is assigned a *profit* π_j^*, this corresponds to looking for a pattern having an overall profit greater than $\sum_{i \leq r} \sigma_i^*$ for some $\kappa_r \in K$. Thus, for a given value $\kappa_r \in K$, our *slave* problem can be formulated as

$$\max \sum_{j=1}^{n} \pi_j^* \xi_j \tag{3.31}$$

$$\sum_{j=1}^{n} w_j \xi_j \leq c \tag{3.32}$$

$$\sum_{j=1}^{n} \xi_j \leq \kappa_r \tag{3.33}$$

$$\xi_j \in \{0, 1\} \quad (j = 1, \ldots, n) \tag{3.34}$$

whose optimal solution identifies the pattern $P \in \mathcal{P}^r(c)$ with maximum profit, namely $P = \{I_j : \xi_j = 1\}$. Problem (3.31)-(3.34) is a *0-1 Knapsack Problem* ((3.31), (3.32), (3.34)) with an additional cardinality constraint. Since the w_j values are non-negative, we can reduce any instance by removing items with non-positive profit. The reduced problem can then be solved to optimality through the algorithm recently presented by Martello and Toth [119] for the *Two-Constraint 0-1 Knapsack Problem*.

In our implementation we start with no variable x_P in the restricted master. At each iteration we add to the restricted master, for each $\kappa_r \in K$, the pattern of $\mathcal{P}^r(c)$ with maximum profit, provided the corresponding constraint (3.28) is violated.

3.5 Computational experiments

The algorithms of the previous sections were coded in C and experimentally tested on a large set of randomly generated instances. The computational experiments were performed on a Pentium III at 1133 Mhz running under a Windows operating system. The LP relaxations produced by the column generation approach for the computation of L_{CG} were solved through CPLEX 7.0.

We generated 81 classes of test problems by combining in all possible ways 9 *weight classes* and 9 *cardinality classes*. The weight classes have been derived from those used by Dell'Amico and Martello [38], and are as follows (See Table 3.1 for the parameters' values.)

Classes $w1$, $w2$ and $w3$: weights w_j uniformly random in $[U_{min}, U_{max}]$.

Classes $w4$, $w5$ and $w6$: weights w_j drawn from an exponential distribution of average value μ, by disregarding non-positive values.

Classes $w7$, $w8$ and $w9$: weights w_j drawn from a normal distribution of average value μ and standard deviation σ, by disregarding non-positive values.

The *cardinality classes* are as follows (See Table 3.2 for the parameters' values.)

Classes $k1, \ldots, k6$: cardinalities k_i uniformly random in $[k_{min}, k_{max}]$

Classes $k7$, $k8$ and $k9$: given a parameter $\delta \geq 1$, k_i values satisfying $\sum_{i=1}^{m} k_i = \lfloor \delta\, n \rfloor$ and $k_i \geq 2\, \forall i$ are generated through the procedure given in Figure 3.4.

Procedure Classes_k7-k9(δ)
1. $Sumk := \lfloor \delta\, n \rfloor - 2\, m$;
2. **for** $i := 1$ **to** $m - 1$ **do**
   $r :=$ random integer in $[0, \lfloor Sumk/2 \rfloor]$;
   $k_i := 2 + r$;
   $Sumk := Sumk - r$;
   endfor
3. $k_m := 2 + Sumk$;
end

Figure 3.4: Data generation for Classes $k7$–$k9$

Uniform			Exponential		Normal		
Class	U_{min}	U_{max}	*Class*	μ	*Class*	μ	σ
$w1$	10	1000	$w4$	25	$w7$	100	33
$w2$	200	1000	$w5$	50	$w8$	100	66
$w3$	500	1000	$w6$	100	$w9$	100	100

Table 3.1: Parameters used for the weight classes

For each generated instance we tested whether conditions $n \leq \sum_{i=1}^{m} k_i$ and $k_i \geq 2\, \forall i$ were satisfied: If not, a new instance was generated in order to avoid infeasible or easily reducible instances. Note that the k_i values of Classes $k1$–$k6$ lay in small ranges, so the corresponding instances usually have

Class	k_{min}	k_{max}	*Class*	k_{min}	k_{max}	*Class*	δ
$k1$	$\lceil n/m \rceil - 1$	$\lceil n/m \rceil$	$k4$	$\lceil n/m \rceil$	$\lceil n/m \rceil + 1$	$k7$	1
$k2$	$\lceil n/m \rceil - 1$	$\lceil n/m \rceil + 1$	$k5$	$\lceil n/m \rceil$	$\lceil n/m \rceil + 2$	$k8$	3/2
$k3$	$\lceil n/m \rceil - 2$	$\lceil n/m \rceil + 2$	$k6$	$\lceil n/m \rceil$	$\lceil n/m \rceil + 3$	$k9$	2

Table 3.2: Parameters used for the cardinality classes

a number of subsets with the same cardinality limit. On the contrary, the k_i values of classes $k7$–$k9$ are usually very sparse. Further observe that instances of Class $k7$ have $n = \sum_{i=1}^{m} k_i$, so each subset S_i must be assigned exactly k_i items. The same may occur with some instances of Classes $k1$–$k6$, in particular with the first three classes.

The algorithms have been tested on random instances with n in {10, 25, 50, 100, 200, 400} and m in {3,4,5,10,20,40,50}. In order to avoid trivial instances, we considered only (n, m) pairs satisfying $n > 2m$, thus obtaining a grand total of 31 pairs. For each quadruple (n, m, w_j class, k_i class) 10 feasible instances were generated, producing 25.110 test problems in total.

Tables 3.3 and 3.4 present the overall performance of lower bounds and heuristic algorithms. In Table 3.3 the entries give, for each weight class, the average performance over all cardinality classes, while in Table 3.4 they give, for each cardinality class, the average performance over all weight classes.

Both Tables 3.3 and 3.4 report the behavior of lower bounds $L_{|\#\leq k_m|}$, L_C (see Section 3.2) and L_{CG} (see Section 3.4), of constructive heuristics *LPT*-k_i, *GH*-k_i and *MS*-k_i (see Section 3.3.1), and of the scatter search algorithm of Section 3.3.2. The last column gives the performance of the overall heuristic, including the possible improvements produced by the column generation computation. The scatter search was executed by receiving in input the best solution found by the constructive heuristics. The tables provide the following information. Let v_L be the value produced by a lower bounding procedure L, and v_H the solution value found by a heuristic algorithm H. Let v_L^* and v_H^* denote the best lower bound and heuristic solution value obtained, respectively. For each lower bound L (resp. heuristic algorithm H) the tables give, for each class

- #best = number of times in which $v_L = v_L^*$ (resp. $v_H = v_H^*$);
- #opt = number of times in which $v_L = v_L^*$ (resp. $v_H = v_H^*$) and $v_L^* = v_H^*$, i.e., a proved optimal value was obtained;
- #missed = number of times in which $v_L < v_L^*$ (resp. $v_H > v_H^*$) and $v_L^* = v_H^*$, i.e., L (resp. H) did not obtain the optimal value, but another lower bound (resp. heuristic) did;
- %gap = average percentage gap. For each instance, the gap was computed as $100(v_L^* - v_L)/v_L^*$ (resp. $100(v_H - v_H^*)/v_L^*$)

The execution time was negligible for the combinatorial lower bounds and the constructive heuristics. The column generation algorithm that produces L_{CG} had a time limit of 60 seconds. The computing times of the complete algorithm (including scatter search and column generation) are reported in Tables 3.6–3.11.

Before discussing the results provided by the tables we recall that lower bound $L_{|\#\leq k_m|}$ is incorporated in bound L_C (see (3.13)) and that the binary search used to compute L_{CG} starts from the best bound previously obtained, namely L_C. Hence L_{CG} dominates L_C which, in turn, dominates $L_{|\#\leq k_m|}$. This can be clearly observed by looking at the columns of Tables 3.3 and 3.4 devoted to the lower bounds. All bounds, however, produce values very close to the optimum (see rows %gap): In particular, for both weight classes and cardinality classes, $L_{|\#\leq k_m|}$ has average gaps around $5 \cdot 10^{-4}$, L_C around $4 \cdot 10^{-4}$ and L_{CG} around $1 \cdot 10^{-4}$.

Concerning the heuristic algorithms, we recall that the scatter search starts from the best solution obtained by the constructive heuristics, although the computational experiments showed that its behaviour do not worsen significantly when started from randomly generated solutions. Looking at

Table 3.3: Overall performance of lower bounds and heuristics on weight classes.

Class		$L_{\lvert\#\leq k_m\rvert}$	L_C	L_{CG}	*LPT-k_i*	*GH-k_i*	*MS-k_i*	*Scatter*	*Overall*
	#best	2596	2607	2790	109	226	1726	2764	2790
$w1$	#opt	2398	2408	2511	109	226	1692	2488	2511
	#missed	113	103	0	2402	2285	819	23	0
	%gap	0.05	0.04	0.01	3.22	1.12	0.57	0.01	0.01
	#best	2605	2615	2790	100	225	1741	2760	2790
$w2$	#opt	2393	2402	2509	100	225	1709	2482	2509
	#missed	116	107	0	2409	2284	800	27	0
	%gap	0.05	0.04	0.01	3.21	1.14	0.57	0.01	0.01
	#best	2609	2617	2790	123	239	1789	2767	2790
$w3$	#opt	2407	2415	2516	123	239	1752	2495	2516
	#missed	109	101	0	2393	2277	764	21	0
	%gap	0.09	0.04	0.01	3.15	1.10	0.56	0.01	0.01
	#best	2599	2605	2790	112	212	1746	2757	2790
$w4$	#opt	2370	2374	2485	112	210	1702	2457	2485
	#missed	115	111	0	2373	2275	783	28	0
	%gap	0.04	0.04	0.01	3.08	1.15	0.61	0.01	0.01
	#best	2618	2625	2790	132	253	1780	2766	2790
$w5$	#opt	2399	2406	2501	132	252	1742	2478	2501
	#missed	102	95	0	2369	2249	759	23	0
	%gap	0.05	0.04	0.01	3.17	1.13	0.56	0.01	0.01
	#best	2610	2617	2790	124	232	1737	2765	2790
$w6$	#opt	2396	2402	2501	124	232	1699	2480	2501
	#missed	105	99	0	2377	2269	802	21	0
	%gap	0.05	0.04	0.01	3.18	1.12	0.58	0.01	0.01
	#best	2613	2622	2790	106	226	1734	2767	2790
$w7$	#opt	2404	2413	2514	106	226	1690	2493	2514
	#missed	110	101	0	2408	2288	824	21	0
	%gap	0.05	0.04	0.01	3.25	1.13	0.57	0.01	0.01
	#best	2605	2614	2790	111	224	1711	2767	2790
$w8$	#opt	2393	2400	2496	111	224	1674	2477	2496
	#missed	103	96	0	2385	2272	822	19	0
	%gap	0.05	0.04	0.01	3.21	1.15	0.60	0.01	0.01
	#best	2628	2633	2790	119	236	1744	2767	2790
$w9$	#opt	2407	2412	2498	119	236	1703	2478	2498
	#missed	91	86	0	2379	2262	795	20	0
	%gap	0.04	0.04	0.01	3.06	1.08	0.55	0.01	0.01
	#best	2609.2	2617.2	2790.0	115.1	230.3	1745.3	2764.4	2790.0
average	#opt	2396.3	2403.5	2503.4	115.1	230.0	1707.0	2480.8	2503.4
	#missed	107.1	99.8	0.0	2388.3	2273.4	796.4	22.5	0.0
	%gap	0.05	0.04	0.01	3.17	1.12	0.58	0.01	0.01

Table 3.4: Overall performance of lower bounds and heuristics on cardinality classes.

Class		$L_{\lvert\#\leq k_m\rvert}$	L_C	L_{CG}	*LPT-k_i*	*GH-k_i*	*MS-k_i*	*Scatter*	*Overall*
	#best	2628	2628	2790	45	79	2229	2759	2790
$k1$	#opt	2624	2624	2724	45	79	2227	2695	2724
	#missed	100	100	0	2679	2645	497	29	0
	%gap	0.08	0.04	0.01	1.50	1.27	0.44	0.01	0.01
	#best	2643	2643	2790	57	122	802	2762	2790
$k2$	#opt	2351	2351	2439	57	122	800	2414	2439
	#missed	88	88	0	2382	2317	1639	25	0
	%gap	0.04	0.04	0.01	1.70	1.31	0.96	0.01	0.01
	#best	2560	2561	2790	42	98	640	2761	2790
$k3$	#opt	2198	2199	2304	42	98	635	2277	2304
	#missed	106	105	0	2262	2206	1669	27	0
	%gap	0.05	0.04	0.01	2.59	1.78	1.30	0.02	0.01
	#best	2658	2658	2790	104	194	2066	2772	2790
$k4$	#opt	2347	2347	2422	104	194	1956	2407	2422
	#missed	75	75	0	2318	2228	466	15	0
	%gap	0.04	0.04	0.01	1.25	0.80	0.29	0.01	0.01
	#best	2650	2650	2790	117	213	2154	2766	2790
$k5$	#opt	2415	2415	2487	117	213	2057	2470	2487
	#missed	72	72	0	2370	2274	430	17	0
	%gap	0.04	0.04	0.01	1.19	0.74	0.19	0.01	0.01
	#best	2653	2653	2790	98	211	2335	2766	2790
$k6$	#opt	2500	2500	2580	98	211	2245	2558	2580
	#missed	80	80	0	2482	2369	335	22	0
	%gap	0.03	0.03	0.01	1.22	0.71	0.12	0.01	0.01
	#best	2475	2510	2790	301	406	943	2750	2790
$k7$	#opt	2321	2354	2547	301	403	931	2510	2547
	#missed	226	193	0	2246	2144	1616	37	0
	%gap	0.11	0.06	0.01	9.66	2.28	1.26	0.01	0.01
	#best	2605	2628	2790	156	420	2227	2778	2790
$k8$	#opt	2412	2431	2519	156	420	2216	2510	2519
	#missed	107	88	0	2363	2099	303	9	0
	%gap	0.05	0.03	0.01	5.23	0.64	0.34	0.01	0.01
	#best	2611	2624	2790	116	330	2312	2766	2790
$k9$	#opt	2399	2411	2509	116	330	2296	2487	2509
	#missed	110	98	0	2393	2179	213	22	0
	%gap	0.05	0.04	0.01	4.20	0.58	0.28	0.01	0.01
	#best	2609.2	2617.2	2790.0	115.1	230.3	1745.3	2764.4	2790.0
average	#opt	2396.3	2403.5	2503.4	115.1	230.0	1707.0	2480.8	2503.4
	#missed	107.1	99.8	0.0	2388.3	2273.4	796.4	22.5	0.0
	%gap	0.05	0.04	0.01	3.17	1.12	0.58	0.01	0.01

the last columns of Tables 3.3 and 3.4, in rows #best and %gap, we see that *LPT-k_i* and *GH-k_i* are outperformed by *MS-k_i* which, in turn, produces solutions largely worse than that of the scatter search. Further improvements are produced by the feasible solutions detected by the column generation algorithm. The best constructive heuristic, *MS-k_i*, produce solutions with gaps aroun $5 \cdot 10^{-3}$ with a maximum of $1.3 \cdot 10^{-2}$, whereas the scatter search has always solutions as close to the lower bound as $1 \cdot 10^{-4}$. On the other hand, the scatter search may require computing times up to 16 minutes (see Tables 3.6-3.11). It is worth noting that the results grouped by weight class (Table 3.3) do not present relevant differences among the different classes. On the contrary the cardinality class (see Table 3.4) determines considerably different behaviours.

Heuristic *LPT-k_i* has average gaps close to $3 \cdot 10^{-2}$, but for classes $k7$–$k9$ where its performance deteriorates. Quite curiously, it found its largest number of best solution exactly in class $k7$, where it presents the largest percentage error.

Algorithm *GH-k_i* roughly has twice the number of best solutions with respect to *LPT-k_i*. The average gaps also improve from around $3 \cdot 10^{-2}$ to around $1 \cdot 10^{-2}$. For this algorithm too the worst gap result ($2.28 \cdot 10^{-2}$) is encountered for Class $k7$.

Heuristic *MS-k_i* has very good performances for almost all classes. The worse gaps are observed for classes $k2$, $k3$ and $k7$.

Finally, the scatter search algorithm is very robust and its performance is excellent on all classes: Its percentage gap is in practice $1 \cdot 10^{-4}$ on all instances. We additionally notice that the best solution for about 1% of the instances was been determined during the computation of the column generation lower bound L_{CG} (see columns *Scatter* and *Overall*, rows #opt and #missed), which increased by about 200 units the number of instances solved to optimality.

Table 3.5: Performance of the reduction procedure.

Weight class	%active	%n-red.	%m-red.	Cardinality class	%active	%n-red.	%m-red.
$w1$	20.68	4.33	9.37	$k1$	0.07	0.01	0.02
$w2$	19.96	4.26	9.17	$k2$	0.04	0.01	0.01
$w3$	20.22	4.27	9.27	$k3$	15.13	1.46	3.26
$w4$	21.25	4.47	9.62	$k4$	0.00	0.00	0.00
$w5$	20.36	4.30	9.33	$k5$	0.00	0.00	0.00
$w6$	20.18	4.32	9.34	$k6$	0.00	0.00	0.00
$w7$	20.47	4.19	9.11	$k7$	66.95	18.12	36.79
$w8$	19.89	4.30	9.30	$k8$	53.98	10.78	24.33
$w9$	20.54	4.34	9.41	$k9$	47.38	8.40	19.50
average	20.39	4.31	9.32	average	20.39	4.31	9.32

In Table 3.5 we give the performance of procedure Reduction2. (As already observed, procedure Reduction1 cannot operate on our data sets.) For each weight class (resp. cardinality class) the table provides the following information, computed over all cardinality classes (resp. weight classes):

- %active = average percentage of instances where some reduction was obtained;
- %n-red = overall percentage of items reduction;
- %m-red = and of subsets reduction.

Again, if we consider the results grouped by weight class we do not see considerable differences, while the results grouped by cardinality class show very small reductions for classes $k1$–$k3$, no reduction at all for classes $k4$–$k6$ and good behavior for classes $k7$–$k9$. For the latter classes, both the reduction of the number of items and of the number of subsets have relevant impact on the final dimension of the instance to be solved, probably due to the fact that these instances are characterized by k_i values with relatively high variance.

Tables 3.6–3.8 and 3.9–3.11 give more detailed results on the performance of the scatter search algorithm. For each feasible pair (n, m), the entries in Tables 3.6–3.8 (resp. in Tables 3.9–3.11) give

Table 3.6: Results for Classes $w1$–$w3$ (uniform distribution)

		$w1 : p_j \in [10, 1000]$					$w2 : p_j \in [200, 1000]$					$w3 : p_j \in [500, 1000]$				
n	m	$\%opt$	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M
10	3	100.00	0.00	0.00	0.03	0.22	100.00	0.00	0.00	0.04	0.28	100.00	0.00	0.00	0.03	0.33
25	3	100.00	0.00	0.00	0.02	1.26	98.89	0.00	0.05	0.02	1.04	100.00	0.00	0.00	0.02	1.54
50	3	98.89	0.00	0.01	0.06	4.95	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.06
100	3	100.00	0.00	0.00	0.01	0.06	98.89	0.00	0.02	0.69	60.80	100.00	0.00	0.00	0.01	0.06
200	3	100.00	0.00	0.00	0.04	1.04	100.00	0.00	0.00	0.04	1.26	100.00	0.00	0.00	0.02	0.11
400	3	100.00	0.00	0.00	0.08	0.33	100.00	0.00	0.00	0.10	0.27	100.00	0.00	0.00	0.11	0.39
10	4	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.02	0.22	100.00	0.00	0.00	0.01	0.06
25	4	97.78	0.00	0.10	0.04	0.82	98.89	0.00	0.03	0.08	0.99	96.67	0.00	0.15	0.05	0.94
50	4	100.00	0.00	0.00	0.01	0.11	98.89	0.00	0.05	0.07	5.49	98.89	0.00	0.02	0.11	8.90
100	4	100.00	0.00	0.00	0.02	0.22	100.00	0.00	0.00	0.02	0.28	100.00	0.00	0.00	0.02	0.55
200	4	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.05	1.81	100.00	0.00	0.00	0.03	0.11
400	4	100.00	0.00	0.00	0.10	0.55	100.00	0.00	0.00	0.10	1.32	100.00	0.00	0.00	0.07	0.33
25	5	88.89	0.00	0.06	0.20	1.37	78.89	0.02	0.60	0.31	1.26	87.78	0.01	0.24	0.25	1.10
50	5	98.89	0.00	0.07	0.10	5.87	98.89	0.00	0.08	0.07	4.34	97.78	0.00	0.11	0.16	6.21
100	5	100.00	0.00	0.00	0.05	0.66	100.00	0.00	0.00	0.03	0.55	100.00	0.00	0.00	0.04	0.66
200	5	100.00	0.00	0.00	0.06	1.27	100.00	0.00	0.00	0.08	1.43	100.00	0.00	0.00	0.04	0.16
400	5	100.00	0.00	0.00	0.11	0.50	100.00	0.00	0.00	0.12	0.77	100.00	0.00	0.00	0.10	0.77
25	10	98.89	0.00	0.09	0.57	2.69	100.00	0.00	0.00	0.53	3.02	100.00	0.00	0.00	0.49	2.58
50	10	66.67	0.01	0.05	2.25	7.53	58.89	0.02	0.05	2.39	7.25	66.67	0.01	0.08	2.34	13.73
100	10	100.00	0.00	0.00	0.40	3.90	100.00	0.00	0.00	0.30	6.15	95.56	0.00	0.03	1.11	26.48
200	10	100.00	0.00	0.00	0.50	11.32	100.00	0.00	0.00	0.38	11.09	100.00	0.00	0.00	0.12	2.47
400	10	100.00	0.00	0.00	0.19	0.82	100.00	0.00	0.00	0.99	70.14	100.00	0.00	0.00	0.27	2.64
50	20	87.78	0.02	0.50	10.45	20.93	92.22	0.01	0.17	9.34	22.19	90.00	0.01	0.17	9.93	24.05
100	20	53.33	0.02	0.09	13.62	47.40	53.33	0.02	0.08	11.80	55.75	55.56	0.02	0.18	10.90	46.85
200	20	96.67	0.00	0.02	3.55	98.26	96.67	0.00	0.02	3.61	96.78	96.67	0.00	0.02	3.16	63.71
400	20	100.00	0.00	0.00	0.45	1.92	100.00	0.00	0.00	2.31	172.30	100.00	0.00	0.00	0.44	2.75
100	40	22.22	0.18	0.47	77.07	245.35	36.67	0.17	0.64	66.80	289.68	30.00	0.18	0.59	68.67	220.26
200	40	61.11	0.02	0.04	45.04	146.54	55.56	0.02	0.08	48.28	159.23	64.44	0.01	0.08	38.20	161.86
400	40	94.44	0.00	0.02	11.87	183.45	94.44	0.00	0.02	10.63	167.20	94.44	0.00	0.02	20.18	612.92
200	50	34.44	0.04	0.11	77.50	337.95	36.67	0.04	0.11	78.27	352.07	32.22	0.04	0.14	82.32	476.20
400	50	90.00	0.00	0.03	32.33	732.43	90.00	0.00	0.03	30.04	474.94	88.89	0.00	0.03	24.36	409.85
summary		90.00	0.01	0.50	8.93	732.43	89.93	0.01	0.64	8.63	474.94	90.18	0.01	0.59	8.50	612.92

Table 3.7: Results for Classes $w4$–$w6$ (exponential distribution)

		$w4:\mu=25$					$w5:\mu=50$					$w6:\mu=100$				
n	m	$\%opt$	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M
10	3	100.00	0.00	0.00	0.03	0.28	100.00	0.00	0.00	0.03	0.23	100.00	0.00	0.00	0.04	0.33
25	3	100.00	0.00	0.00	0.08	2.25	98.89	0.00	0.02	0.06	2.58	98.89	0.00	0.02	0.03	1.26
50	3	98.89	0.00	0.02	0.04	3.40	100.00	0.00	0.00	0.00	0.05	98.89	0.00	0.06	0.06	4.67
100	3	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.02	0.39	100.00	0.00	0.00	0.01	0.06
200	3	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.03	0.50	100.00	0.00	0.00	0.06	2.52
400	3	100.00	0.00	0.00	0.09	0.28	100.00	0.00	0.00	0.08	0.28	100.00	0.00	0.00	0.26	15.27
10	4	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.02	0.27	100.00	0.00	0.00	0.02	0.28
25	4	98.89	0.00	0.09	0.05	1.53	97.78	0.00	0.04	0.06	1.21	97.78	0.00	0.19	0.04	1.04
50	4	98.89	0.00	0.03	0.05	4.06	97.78	0.00	0.03	0.08	4.01	98.89	0.00	0.05	0.07	5.05
100	4	100.00	0.00	0.00	0.02	0.33	100.00	0.00	0.00	0.02	0.17	100.00	0.00	0.00	0.01	0.16
200	4	100.00	0.00	0.00	0.04	0.28	100.00	0.00	0.00	0.04	0.33	100.00	0.00	0.00	0.07	3.57
400	4	100.00	0.00	0.00	0.08	0.33	100.00	0.00	0.00	0.08	0.55	100.00	0.00	0.00	0.09	0.44
25	5	91.11	0.00	0.05	0.24	1.21	82.22	0.01	0.05	0.41	1.59	93.33	0.01	0.27	0.20	1.20
50	5	98.89	0.00	0.04	0.10	7.08	97.78	0.00	0.07	0.14	6.75	98.89	0.00	0.05	0.06	3.52
100	5	98.89	0.00	0.01	0.10	5.98	98.89	0.00	0.01	0.28	20.65	100.00	0.00	0.00	0.03	0.39
200	5	100.00	0.00	0.00	0.04	0.39	100.00	0.00	0.00	0.09	2.63	100.00	0.00	0.00	0.11	3.84
400	5	100.00	0.00	0.00	0.11	0.55	100.00	0.00	0.00	0.12	0.60	100.00	0.00	0.00	0.55	41.36
25	10	100.00	0.00	0.00	0.43	2.52	100.00	0.00	0.00	0.36	3.07	100.00	0.00	0.00	0.47	2.75
50	10	53.33	0.02	0.09	3.23	35.65	60.00	0.02	0.09	2.78	9.72	57.78	0.02	0.05	2.55	9.39
100	10	98.89	0.00	0.02	0.61	18.07	100.00	0.00	0.00	0.32	4.62	95.56	0.00	0.04	1.01	22.20
200	10	100.00	0.00	0.00	0.08	0.34	100.00	0.00	0.00	0.25	12.14	100.00	0.00	0.00	0.35	10.66
400	10	100.00	0.00	0.00	0.72	41.96	100.00	0.00	0.00	0.23	0.88	100.00	0.00	0.00	0.20	0.93
50	20	88.89	0.01	0.18	11.47	25.70	88.89	0.02	0.45	11.21	71.84	86.67	0.01	0.22	10.96	21.64
100	20	48.89	0.02	0.11	14.61	87.55	55.56	0.02	0.12	12.75	74.70	43.33	0.02	0.09	13.84	37.90
200	20	92.22	0.00	0.02	7.62	103.48	90.00	0.00	0.02	7.53	99.86	96.67	0.00	0.02	3.84	111.45
400	20	100.00	0.00	0.00	0.48	2.42	100.00	0.00	0.00	2.35	173.18	100.00	0.00	0.00	8.14	177.08
100	40	32.22	0.17	0.62	71.86	279.02	37.78	0.17	0.71	62.57	249.19	30.00	0.18	1.04	68.80	287.42
200	40	53.33	0.02	0.16	48.73	142.14	61.11	0.02	0.04	43.46	219.38	65.56	0.01	0.04	38.96	154.02
400	40	93.33	0.00	0.02	16.44	386.68	94.44	0.00	0.02	16.22	603.68	97.78	0.00	0.02	5.34	146.09
200	50	25.56	0.05	0.14	83.78	503.06	27.78	0.05	0.16	84.71	398.87	30.00	0.04	0.10	91.93	1019.63
400	50	88.89	0.00	0.03	35.04	492.74	90.00	0.00	0.03	25.04	484.34	88.89	0.00	0.03	21.70	200.97
summary		89.07	0.01	0.62	9.56	503.06	89.64	0.01	0.71	8.75	603.68	89.64	0.01	1.04	8.70	1019.63

Table 3.8: Results for Classes $w7$–$w9$ (normal distribution)

		$w7: \mu = 100, \sigma = 33$					$w8: \mu = 100, \sigma = 66$					$w9: \mu = 100, \sigma = 100$				
n	m	$\%opt$	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M
10	3	100.00	0.00	0.00	0.03	0.33	100.00	0.00	0.00	0.03	0.28	100.00	0.00	0.00	0.03	0.38
25	3	98.89	0.00	0.12	0.05	2.27	98.89	0.00	0.12	0.06	2.03	100.00	0.00	0.00	0.06	2.19
50	3	100.00	0.00	0.00	0.00	0.06	98.89	0.00	0.01	0.05	4.39	100.00	0.00	0.00	0.00	0.06
100	3	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.02	0.39
200	3	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.02	0.11
400	3	100.00	0.00	0.00	0.12	0.34	100.00	0.00	0.00	0.11	0.50	100.00	0.00	0.00	0.09	0.28
10	4	100.00	0.00	0.00	0.01	0.22	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.22
25	4	98.89	0.00	0.03	0.07	1.09	98.89	0.00	0.10	0.04	0.88	97.78	0.00	0.10	0.04	0.77
50	4	98.89	0.00	0.06	0.07	5.61	98.89	0.00	0.09	0.05	3.02	100.00	0.00	0.00	0.01	0.11
100	4	100.00	0.00	0.00	0.03	0.39	100.00	0.00	0.00	0.03	0.44	100.00	0.00	0.00	0.02	0.28
200	4	100.00	0.00	0.00	0.04	0.55	100.00	0.00	0.00	0.04	0.22	100.00	0.00	0.00	0.07	1.92
400	4	100.00	0.00	0.00	0.09	0.39	100.00	0.00	0.00	0.08	0.50	100.00	0.00	0.00	0.08	0.55
25	5	84.44	0.02	0.71	0.26	1.31	85.56	0.01	0.25	0.26	1.11	85.56	0.01	0.05	0.30	1.14
50	5	100.00	0.00	0.00	0.21	17.19	100.00	0.00	0.00	0.03	0.33	100.00	0.00	0.00	0.02	0.17
100	5	100.00	0.00	0.00	0.02	0.38	100.00	0.00	0.00	0.05	0.70	100.00	0.00	0.00	0.06	0.67
200	5	100.00	0.00	0.00	0.04	0.22	100.00	0.00	0.00	0.05	0.17	100.00	0.00	0.00	0.09	3.14
400	5	100.00	0.00	0.00	0.15	2.53	100.00	0.00	0.00	0.12	0.77	100.00	0.00	0.00	0.21	9.22
25	10	100.00	0.00	0.00	0.53	3.14	100.00	0.00	0.00	0.46	2.63	100.00	0.00	0.00	0.44	3.34
50	10	63.33	0.02	0.08	2.23	18.08	71.11	0.01	0.08	2.01	9.94	62.22	0.01	0.05	2.34	7.64
100	10	98.89	0.00	0.02	0.55	19.13	96.67	0.00	0.02	0.92	17.47	96.67	0.00	0.04	1.23	34.94
200	10	100.00	0.00	0.00	0.17	3.36	100.00	0.00	0.00	0.10	0.81	100.00	0.00	0.00	0.21	10.66
400	10	100.00	0.00	0.00	0.51	18.23	100.00	0.00	0.00	0.21	0.98	100.00	0.00	0.00	0.68	40.86
50	20	88.89	0.01	0.25	9.86	22.97	86.67	0.02	0.24	11.61	26.42	90.00	0.01	0.13	9.08	27.02
100	20	48.89	0.02	0.09	12.60	57.61	47.78	0.02	0.08	13.71	74.25	54.44	0.02	0.09	12.61	83.05
200	20	94.44	0.00	0.02	6.76	100.78	92.22	0.00	0.02	7.01	98.42	95.56	0.00	0.02	4.58	71.45
400	20	100.00	0.00	0.00	0.40	1.81	100.00	0.00	0.00	2.45	180.44	100.00	0.00	0.00	4.26	174.05
100	40	33.33	0.18	0.73	66.33	444.02	26.67	0.19	0.66	71.50	223.22	32.22	0.16	0.48	69.51	370.64
200	40	64.44	0.01	0.04	41.01	217.63	58.89	0.02	0.09	50.04	216.41	54.44	0.02	0.04	48.51	135.72
400	40	92.22	0.00	0.02	18.95	641.03	94.44	0.00	0.02	21.78	506.64	94.44	0.00	0.02	10.64	170.33
200	50	34.44	0.04	0.12	83.51	344.06	25.56	0.05	0.11	91.68	408.48	26.67	0.04	0.10	85.61	529.31
400	50	93.33	0.00	0.03	22.23	607.31	92.22	0.00	0.03	17.53	197.80	85.56	0.00	0.03	36.33	734.56
summary		90.11	0.01	0.73	8.61	641.03	89.46	0.01	0.66	9.42	506.64	89.53	0.01	0.48	9.26	734.56

Table 3.9: Results for Classes $k1$–$k3$

		$k1$					$k2$					$k3$				
n	m	$\%opt$	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M
10	3	100.00	0.00	0.00	0.04	0.33	100.00	0.00	0.00	0.02	0.22	100.00	0.00	0.00	0.05	0.27
25	3	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.06	97.78	0.00	0.12	0.04	2.03
50	3	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.00	0.06
100	3	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.02	0.06
200	3	100.00	0.00	0.00	0.04	0.11	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.05	0.11
400	3	100.00	0.00	0.00	0.18	0.34	100.00	0.00	0.00	0.13	0.34	100.00	0.00	0.00	0.14	0.28
10	4	100.00	0.00	0.00	0.01	0.22	100.00	0.00	0.00	0.02	0.27	100.00	0.00	0.00	0.03	0.28
25	4	100.00	0.00	0.00	0.01	0.17	98.89	0.00	0.10	0.02	0.55	100.00	0.00	0.00	0.01	0.06
50	4	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.01	0.11	100.00	0.00	0.00	0.02	0.11
100	4	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.04	0.44	100.00	0.00	0.00	0.03	0.22
200	4	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.06	0.22	100.00	0.00	0.00	0.07	0.17
400	4	100.00	0.00	0.00	0.05	0.22	100.00	0.00	0.00	0.17	0.39	100.00	0.00	0.00	0.17	0.50
25	5	98.89	0.00	0.04	0.18	1.10	87.78	0.02	0.71	0.26	1.14	81.11	0.02	0.27	0.41	1.44
50	5	100.00	0.00	0.00	0.02	0.28	100.00	0.00	0.00	0.06	0.49	100.00	0.00	0.00	0.04	0.55
100	5	100.00	0.00	0.00	0.02	0.11	100.00	0.00	0.00	0.09	0.72	100.00	0.00	0.00	0.08	0.67
200	5	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.09	0.66	100.00	0.00	0.00	0.08	0.17
400	5	100.00	0.00	0.00	0.05	0.22	100.00	0.00	0.00	0.23	0.55	100.00	0.00	0.00	0.24	0.60
25	10	100.00	0.00	0.00	0.42	2.58	100.00	0.00	0.00	0.35	2.08	98.89	0.00	0.09	1.04	3.34
50	10	100.00	0.00	0.00	0.11	2.70	45.56	0.02	0.05	3.14	7.36	38.89	0.03	0.09	3.74	12.25
100	10	100.00	0.00	0.00	0.11	5.33	97.78	0.00	0.02	1.38	34.94	96.67	0.00	0.02	1.35	19.13
200	10	100.00	0.00	0.00	0.07	0.23	100.00	0.00	0.00	0.54	10.93	100.00	0.00	0.00	0.91	12.14
400	10	100.00	0.00	0.00	0.12	0.39	100.00	0.00	0.00	1.41	41.96	100.00	0.00	0.00	0.49	0.99
50	20	93.33	0.01	0.43	8.86	50.42	92.22	0.01	0.50	10.15	71.84	46.67	0.06	0.45	15.48	27.02
100	20	100.00	0.00	0.00	0.19	2.42	37.78	0.03	0.08	19.57	83.05	8.89	0.04	0.18	24.31	87.55
200	20	100.00	0.00	0.00	0.15	0.32	86.67	0.00	0.02	14.19	99.86	92.22	0.00	0.02	9.47	111.45
400	20	100.00	0.00	0.00	0.23	0.44	100.00	0.00	0.00	12.70	180.44	100.00	0.00	0.00	6.96	177.08
100	40	35.56	0.18	0.73	59.02	126.16	41.11	0.15	0.46	54.09	93.76	2.22	0.23	0.41	164.29	444.02
200	40	100.00	0.00	0.00	3.92	219.38	28.89	0.03	0.04	90.78	178.30	21.11	0.03	0.08	96.64	217.63
400	40	100.00	0.00	0.00	0.89	10.20	92.22	0.00	0.02	26.16	603.68	84.44	0.00	0.02	53.02	641.03
200	50	98.89	0.00	0.05	6.66	529.31	15.56	0.05	0.14	109.25	176.26	8.89	0.05	0.16	205.60	1019.63
400	50	100.00	0.00	0.00	1.88	16.69	85.56	0.00	0.03	78.27	734.56	82.22	0.00	0.03	46.70	607.31
summary		97.63	0.01	0.73	2.69	529.31	87.42	0.01	0.71	13.65	734.56	82.58	0.01	0.45	20.37	1019.63

Table 3.10: Results for Classes $k4$–$k6$

		$k4$					$k5$					$k6$				
n	m	$\%opt$	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M
10	3	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.02	0.06
25	3	100.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
50	3	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.06
100	3	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06
200	3	100.00	0.00	0.00	0.03	0.06	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.03	0.11
400	3	100.00	0.00	0.00	0.09	0.17	100.00	0.00	0.00	0.10	0.28	100.00	0.00	0.00	0.09	0.17
10	4	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06
25	4	100.00	0.00	0.00	0.00	0.11	100.00	0.00	0.00	0.00	0.11	100.00	0.00	0.00	0.00	0.11
50	4	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.05
100	4	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.02	0.06
200	4	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.03	0.11	100.00	0.00	0.00	0.03	0.11
400	4	100.00	0.00	0.00	0.05	0.22	100.00	0.00	0.00	0.04	0.16	100.00	0.00	0.00	0.05	0.22
25	5	88.89	0.00	0.04	0.18	0.94	82.22	0.01	0.05	0.24	1.11	94.44	0.00	0.04	0.16	1.09
50	5	100.00	0.00	0.00	0.02	0.11	100.00	0.00	0.00	0.01	0.33	100.00	0.00	0.00	0.01	0.27
100	5	100.00	0.00	0.00	0.02	0.11	100.00	0.00	0.00	0.03	0.06	100.00	0.00	0.00	0.02	0.06
200	5	100.00	0.00	0.00	0.04	0.11	100.00	0.00	0.00	0.04	0.22	100.00	0.00	0.00	0.04	0.13
400	5	100.00	0.00	0.00	0.07	0.28	100.00	0.00	0.00	0.06	0.28	100.00	0.00	0.00	0.05	0.22
25	10	100.00	0.00	0.00	0.32	2.36	100.00	0.00	0.00	0.45	2.69	100.00	0.00	0.00	0.39	3.02
50	10	50.00	0.02	0.08	2.81	7.74	67.78	0.01	0.05	1.95	6.98	83.33	0.01	0.05	1.18	7.41
100	10	97.78	0.00	0.02	0.95	26.48	98.89	0.00	0.02	0.66	23.52	97.78	0.00	0.02	0.85	25.82
200	10	100.00	0.00	0.00	0.09	0.33	100.00	0.00	0.00	0.09	0.33	100.00	0.00	0.00	0.09	0.28
400	10	100.00	0.00	0.00	0.14	0.55	100.00	0.00	0.00	0.14	0.66	100.00	0.00	0.00	0.13	0.55
50	20	97.78	0.00	0.17	9.83	19.06	92.22	0.01	0.19	9.98	22.97	95.56	0.00	0.17	11.57	21.36
100	20	18.89	0.03	0.09	19.16	31.03	52.22	0.02	0.07	11.26	28.39	82.22	0.01	0.08	4.33	30.20
200	20	93.33	0.00	0.02	5.54	64.48	90.00	0.00	0.02	7.85	89.75	90.00	0.00	0.02	10.00	103.48
400	20	100.00	0.00	0.00	0.36	1.48	100.00	0.00	0.00	0.38	2.30	100.00	0.00	0.00	0.29	1.20
100	40	35.56	0.19	0.59	51.71	97.33	30.00	0.19	0.66	63.86	111.34	38.89	0.16	0.55	60.98	102.81
200	40	26.67	0.03	0.04	82.82	145.39	34.44	0.03	0.04	62.76	132.03	41.11	0.02	0.04	51.31	93.38
400	40	92.22	0.00	0.02	15.29	179.39	86.67	0.00	0.02	25.43	206.31	94.44	0.00	0.02	11.05	176.15
200	50	7.78	0.05	0.11	81.50	159.56	47.78	0.03	0.10	41.32	110.13	72.22	0.02	0.10	21.58	110.40
400	50	82.22	0.00	0.03	35.67	203.39	81.11	0.00	0.03	35.26	195.32	76.67	0.01	0.03	46.65	200.30
summary		86.81	0.01	0.59	9.90	203.39	89.14	0.01	0.66	8.45	206.31	92.47	0.01	0.55	7.13	200.30

Table 3.11: Results for Classes $k7$–$k9$

		$k7$					$k8$					$k9$				
n	m	$\%opt$	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M	opt	$\%gap$	$\%gap_M$	t	t_M
10	3	100.00	0.00	0.00	0.10	0.38	100.00	0.00	0.00	0.02	0.06	100.00	0.00	0.00	0.02	0.06
25	3	96.67	0.00	0.12	0.28	2.58	100.00	0.00	0.00	0.03	0.88	100.00	0.00	0.00	0.06	1.33
50	3	96.67	0.00	0.06	0.15	4.95	100.00	0.00	0.00	0.00	0.00	98.89	0.00	0.01	0.05	4.39
100	3	98.89	0.00	0.02	0.69	60.80	100.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
200	3	100.00	0.00	0.00	0.09	2.52	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.00
400	3	100.00	0.00	0.00	0.31	15.27	100.00	0.00	0.00	0.00	0.11	100.00	0.00	0.00	0.00	0.00
10	4	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06
25	4	84.44	0.01	0.19	0.31	1.53	100.00	0.00	0.00	0.07	0.82	100.00	0.00	0.00	0.05	0.77
50	4	93.33	0.00	0.09	0.28	5.61	98.89	0.00	0.02	0.10	8.90	98.89	0.00	0.05	0.06	5.49
100	4	100.00	0.00	0.00	0.06	0.55	100.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
200	4	100.00	0.00	0.00	0.16	3.57	100.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
400	4	100.00	0.00	0.00	0.24	1.32	100.00	0.00	0.00	0.01	0.22	100.00	0.00	0.00	0.00	0.05
25	5	80.00	0.02	0.60	0.43	1.31	80.00	0.01	0.05	0.26	1.15	84.44	0.01	0.06	0.30	1.59
50	5	93.33	0.00	0.11	0.39	7.08	100.00	0.00	0.00	0.00	0.06	97.78	0.00	0.07	0.32	17.19
100	5	97.78	0.00	0.01	0.41	20.65	100.00	0.00	0.00	0.00	0.05	100.00	0.00	0.00	0.00	0.06
200	5	100.00	0.00	0.00	0.30	3.84	100.00	0.00	0.00	0.00	0.11	100.00	0.00	0.00	0.00	0.05
400	5	100.00	0.00	0.00	0.87	41.36	100.00	0.00	0.00	0.02	0.28	100.00	0.00	0.00	0.00	0.11
25	10	100.00	0.00	0.00	0.50	2.25	100.00	0.00	0.00	0.33	2.52	100.00	0.00	0.00	0.49	2.69
50	10	68.89	0.01	0.08	3.52	35.65	46.67	0.02	0.08	2.98	8.62	58.89	0.02	0.09	2.70	8.63
100	10	97.78	0.00	0.03	0.30	11.37	97.78	0.00	0.04	0.41	17.14	97.78	0.00	0.04	0.44	20.33
200	10	100.00	0.00	0.00	0.35	3.36	100.00	0.00	0.00	0.01	0.17	100.00	0.00	0.00	0.00	0.06
400	10	100.00	0.00	0.00	1.54	70.14	100.00	0.00	0.00	0.03	0.78	100.00	0.00	0.00	0.00	0.06
50	20	94.44	0.01	0.17	9.03	17.25	93.33	0.01	0.17	9.32	18.95	94.44	0.01	0.22	9.71	22.41
100	20	75.56	0.01	0.12	7.99	74.70	51.11	0.02	0.09	13.39	42.13	34.44	0.03	0.09	16.23	42.57
200	20	98.89	0.00	0.01	0.45	34.00	100.00	0.00	0.00	0.01	0.50	100.00	0.00	0.00	0.01	0.06
400	20	100.00	0.00	0.00	0.34	2.75	100.00	0.00	0.00	0.02	0.55	100.00	0.00	0.00	0.00	0.06
100	40	35.56	0.15	0.52	54.15	97.88	32.22	0.18	1.04	54.83	91.83	30.00	0.16	0.44	60.18	99.36
200	40	100.00	0.00	0.00	0.02	0.66	95.56	0.00	0.16	6.39	216.41	91.11	0.00	0.09	7.60	109.85
400	40	100.00	0.00	0.00	0.19	1.75	100.00	0.00	0.00	0.01	0.06	100.00	0.00	0.00	0.01	0.06
200	50	17.78	0.06	0.11	86.44	174.02	3.33	0.07	0.11	103.53	180.38	1.11	0.07	0.12	103.43	212.34
400	50	100.00	0.00	0.00	0.12	1.10	100.00	0.00	0.00	0.02	0.22	100.00	0.00	0.00	0.04	2.80
summary		91.29	0.01	0.60	5.48	174.02	90.29	0.18	0.06	103.53	19.13	89.93	0.01	0.44	6.51	212.34

the values, computed over the 90 instances generated for each cardinality class (resp. weight class), of:

- $\%opt$ = average percentage of proved optimal solutions;
- $\%gap$ = average percentage gap, computed as for Tables 3.3 and 3.4;
- $\%gap_M$ = maximum percentage gap;
- t = average computing time;
- t_M = maximum computing time.

The results in these tables confirm the robustness and stability of the scatter search algorithm, with respect to both variations of weight and cardinality.

The algorithm almost systematically solves to optimality instances up to 400 items and 10 subsets. This behavior deteriorates for larger instances, but the overall performance remains satisfactory: The percentage gap never exceeds 1.04 within reasonable CPU times, that are at most around 1000 seconds on an average speed computer.

It is interesting to observe that instances with 25–50 items are often more difficult to solve than larger instances. This is probably due to the fact that a higher number of items, allowing a much higher number of weight combinations, makes it easier to obtain solutions of value close to that of the lower bound.

Part III

Algorithms for Packing Problems

Chapter 4

Metaheuristics for the Strip Packing Problem

[1]

Given a set of rectangular items and a strip of given width, we consider the problem of allocating all the items to a minimum height strip. We present a Tabu search algorithm, a genetic algorithm and we combine the two into a hybrid approach. The performance of the proposed algorithms is evaluated through extensive computational experiments on instances from the literature and on randomly generated instances.

4.1 Introduction

In the *Two-Dimensional Strip Packing Problem* (2SP) we are given a set of n rectangular *items* $j = 1, \ldots, n$, each having a *width* w_j and *height* h_j, and a *strip* of width W and infinite height. The objective is to find a pattern that allocates all the items to the strip, without overlapping, by minimizing the height at which the strip is used. We assume that the items have fixed orientation, i.e., they have to be packed with their base parallel to the base of the strip. Consider the following numerical example: $n = 6$, $w_1 = 6$, $h_1 = 3$, $w_2 = 5$, $h_2 = 2$, $w_3 = 2$, $h_3 = 4$, $w_4 = 3$, $h_4 = 4$, $w_5 = 3$, $h_5 = 3$, $w_6 = 2$, $h_6 = 1$. Feasible (and optimal) strip packings of height 7 are depicted in Figure 4.1 (b) and (c).

Problem 2SP has several real-world applications: cutting of paper or cloth from standardized rolls, cutting of wood, metal or glass from standardized stock pieces, allocating memory in computers, to mention the most relevant ones.

The problem is related to the *Two-Dimensional Bin Packing Problem* (2BP), in which, instead of the strip, one has an unlimited number of identical rectangular *bins* of width W and height H, and the objective is to allocate all the items to the minimum number of bins. Recent surveys on two-dimensional packing problems have been presented by Lodi, Martello and Vigo [113] and Lodi, Martello and Monaci [108], while Dyckhoff, Scheithauer and Terno [49] have published an annotated bibliography on cutting and packing.

Both 2SP and 2BP are NP-hard in the strong sense. Consider indeed the strongly NP-hard *One-Dimensional Bin Packing Problem* (1BP): partition n elements $j = 1, \ldots, n$, each having a size w_j, into the minimum number of subsets so that the total size in no subset exceeds a given capacity W. It is easily seen that both 2SP and 2BP generalize 1BP.

[1] The results of this chapter appears in: M. Iori, S. Martello and M. Monaci, Metaheuristic Algorithms for the Strip Packing problem, in P. Pardalos and V. Korotkich editors, *Optimization and Industry: New Frontiers*, Kluwer Academic Publisher, 2003 [97].

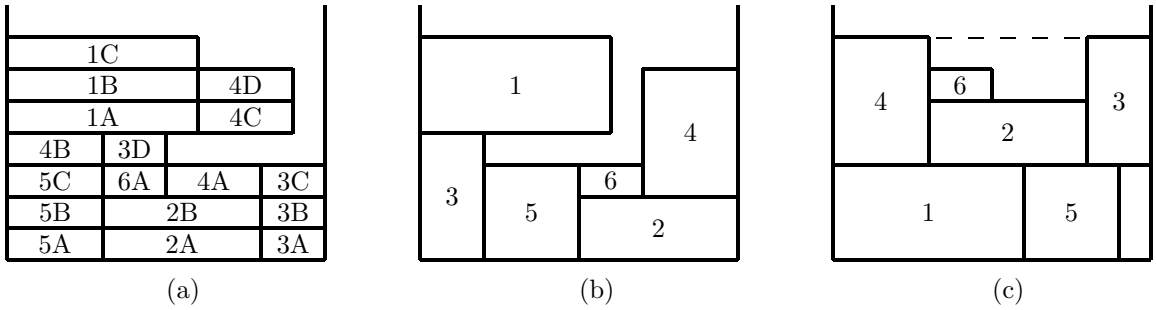

Figure 4.1: (a) optimal solution for the 1BP relaxation; (b) 2SP feasible solution found by algorithm BUILD; (c) 2SP feasible solution found by algorithm $\mathrm{TP}_{\mathrm{2SP}}$.

Approximation algorithms for two-dimensional strip packing problems have been presented by Coffman, Garey, Johnson and Tarjan [27], Baker, Coffman and Rivest [7], Sleator [139], Brown [20], Golan [74], Baker, Brown and Katseff [6], Baker and Schwarz [8], Høyland [95], Jakobs [99] and Steinberg [140]. A general framework for the exact solution of multi-dimensional packing problems has been recently proposed by Fekete and Schepers [52, 51, 53], while lower bounds, approximation algorithms and an exact branch-and-bound approach for 2SP have been given by Martello, Monaci and Vigo [115].

In this paper we examine metaheuristic approaches to 2SP (see, e.g., Reeves [134] and Davis [36] for general introductions to metaheuristics). Section 4.2 deals with lower bounds from the literature and deterministic approximation algorithms. We present a Tabu search algorithm in Section 4.3, and a genetic algorithm in Section 4.4. A hybrid algorithm that combines the two approaches is obtained in Section 4.5. The results of extensive computational tests are finally presented in Section 4.6.

We assume in the following, without loss of generality, that all input data are positive integers, and that $w_j \leq W$ $(j = 1, \ldots, n)$.

4.2 Lower and upper bounds

In this section we discuss deterministic heuristics that are used in the metaheuristic algorithms introduced in the next sections.

Martello, Monaci and Vigo [115] recently proposed the following lower bound for 2SP. Consider a relaxation in which each item j is "cut" into h_j unit-height *slices* of width w_j. The lower bound given by the minimum height of a strip of width W that packs the resulting $\sum_{j=1}^{n} h_j$ slices can then be determined by solving a *One-Dimensional Contiguous Bin Packing Problem* (1CBP), i.e., a 1BP instance having capacity W, with the additional requirement that for each original item j, the h_j unit-height slices of width w_j be packed into h_j contiguous one-dimensional bins. As an example, consider again the instance introduced in Section 4.1: the optimal 1CBP solution is shown in Figure 4.1(a), so a valid lower bound value for the instance is 7.

Given the solution to the 1CBP relaxation, a feasible solution to the original 2SP instance can be obtained through an algorithm, BUILD (see [115]), that re-joins the slices as follows. Consider the first slice of each item, according to non-decreasing one-dimensional bin, breaking ties by decreasing item height. The corresponding two-dimensional item is packed in the lowest position where it fits. It is left (resp. right) justified, if its left (resp. right) edge can touch either the left (resp. right) side of the strip or the right (resp. left) edge of a previous item whose top edge is not lower than that of the current item; otherwise, it is left or right justified, with its edge touching the previous item whose top edge is the tallest one. For the previous example, the items are packed according to the sequence (3, 5, 2, 4, 6, 1), as shown in Figure 4.1(b). Different solutions can be obtained by breaking bin ties

according to different policies: decreasing item width or decreasing item area.

A different heuristic was obtained by adapting to 2SP algorithm TP_{RF} (*Touching Perimeter*), proposed by Lodi, Martello and Vigo [112] for a variant of 2BP. Let L be the lower bound value produced by the 1CBP relaxation. The algorithm, called TP_{2SP}, initializes the strip at height L, and considers the items according to a given ordering. (When executed from scratch, the items are sorted according to non-increasing area, breaking ties by non-increasing $\min\{w_j, h_j\}$ values.) The first item is packed in the bottom-left corner. Let X (resp. Y) denote the set of x-coordinates (resp. y-coordinates) corresponding to corners of already packed items. Each subsequent item is then packed with its bottom-left corner at a coordinate (x, y) ($x \in X$, $y \in Y$) that maximizes the *touching fraction*, i.e., the fraction of the item perimeter that touches either the sides of already packed items or the sides of the strip (including the "ceiling" initially placed at height L). If no feasible packing position exists for an item, the strip ceiling is heightened by packing the item in the position for which the increase in the strip height is a minimum. For the previous example, the items are packed according to the sequence (1, 4, 2, 5, 3, 6), as shown in Figure 4.1(c), where the dashed line represents the ceiling. Different solutions can be obtained by sorting the items according to different strategies.

Our *initialization phase*, common to the three metaheuristic algorithms presented in the next sections, also includes use of a post-optimization procedure (algorithm POST-OPT described in the next section). It can be outlined as follows:

1. compute lower bound L;
2. execute algorithm BUILD, followed by POST-OPT;
3. execute algorithm TP_{2SP}, followed by POST-OPT;
4. select the best solution as the incumbent.

4.3 Tabu search

A Tabu search scheme, 2BP_TS, recently proposed by Lodi, Martello and Vigo [110, 111, 112] proved to be very effective for two-dimensional bin packing problems. It was thus quite natural to use it as a starting point for the strip packing problem.

The core of our approach can be outlined as follows. Let z^* denote the incumbent solution value for 2SP, initially determined, e.g., through algorithms BUILD and TP_{2SP} of Section 4.2. Let S_{2SP} be an initial feasible solution to 2SP, of value not less than z^*. Let $H \geq \max_j\{h_j\}$ be a given threshold value for an instance of 2BP induced by the item set of the 2SP instance, with bin size $W \times H$. The following procedure returns, for the 2SP instance, a new feasible solution of value z, by exploring 2SP solutions derived from 2BP solutions explored by 2BP_TS. A solution to a 2SP instance is described by the coordinates (x_j, y_j) at which the bottom-left corner of item j is placed ($j = 1, \ldots, n$), by assuming a coordinate system with origin in the bottom-left corner of the strip. Given a solution S, we denote by $z(S) = \max_j\{y_j + h_j\}$ the corresponding strip height.

function EXPLORE(S_{2SP},H):
    derive from S_{2SP} a feasible solution S_{2BP} for the induced 2BP instance;
    execute 2BP_TS starting from S_{2BP};
    let Σ_{2BP} be the set of explored 2BP solutions;
    $z := \infty$;
    for each $\overline{S}_{2BP} \in \Sigma_{2BP}$ **do**
        derive from $\overline{S}_{2BP}$ a feasible solution $\overline{S}_{2SP}$ for the 2SP instance;
        apply a post-optimization algorithm to $\overline{S}_{2SP}$;
        $z := \min(z, z(\overline{S}_{2SP}))$
    end for
    return z
end.

We next detail the main steps of EXPLORE. Initially, given an S_{2SP} strip packing, we derive a feasible solution for the induced bin packing instance as follows. The given strip is "cut" into $m = \lceil z(S_{2SP})/H \rceil$ bins, starting from the bottom. For each resulting bin i ($i = 1, \ldots, m-1$), each item crossing the border that separates i from $i+1$ is assigned to a new bin, while the items entirely packed within bin i (if any) remain assigned to i. The whole operation requires $O(n)$ time since, for each item j, we can establish in constant time whether it crosses the border at height $(\lfloor y_j/H \rfloor + 1)H$. The resulting bin packing is clearly not optimized. The reason for this choice is that the computational experiments performed on 2BP_TS (see [112]) showed that the algorithm has the best performance when initialized with solutions very far from the optimum.

In the "for each" loop of EXPLORE, we first derive, from a bin packing, a feasible strip packing as follows. We number the bins used in the packing as $1, \ldots, m$, and initially construct a strip of height mH by simply placing the base of bin i at height $(i-1)H$. The resulting strip packing is then improved through item shifting, by considering the items according to non-decreasing y_j value: the current item is shifted down and then left as much as possible. This step can be performed in $O(n^2)$ time by examining, for each item j, the items k with $y_k + h_k \leq y_j$ and establishing, in constant time, whether intervals $(x_j, x_j + w_j)$ and $(x_k, x_k + w_k)$ intersect. (A similar operation is then executed to shift left item j.) A second solution is obtained by considering the bins according to their filling (through the *filling function* introduced in Lodi, Martello and Vigo [112]), and constructing a strip as follows. We start by placing at the bottom the most filled bin, then we add the least filled one in bottom-up position (i.e., rotated by 180 degrees) then the second most filled one, and so on. The best of the two obtained solutions is finally selected.

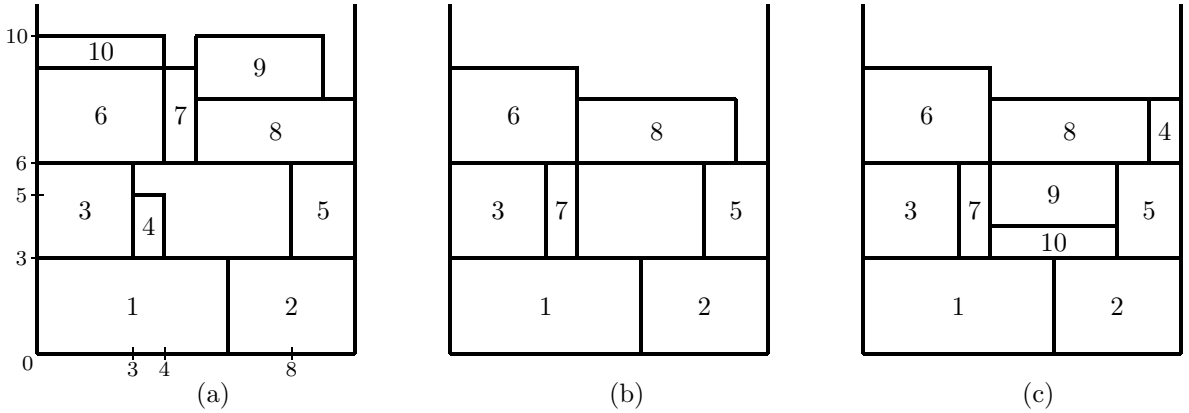

Figure 4.2: (a) initial solution; (b) partial solution after step 4; (c) final solution.

The resulting strip packing is further improved by a post-optimization algorithm, POST-OPT, that iteratively executes the following five steps (illustrated through the example in Figure 4.2).

1. Determine *holes* in the packing, i.e., inner rectangles $[\underline{x}, \underline{y}, \overline{x}, \overline{y}]$ (bottom-left corner in $(\underline{x}, \underline{y})$, top-right corner in $(\overline{x}, \overline{y})$) that contain no (part of) item. In Figure 4.2(a), we have holes $[4, 3, 8, 6]$ and $[3, 5, 8, 6]$.
2. Consider the pairs of holes $([\underline{x}, \underline{y}, \overline{x}, \overline{y}], [\underline{x}', \underline{y}', \overline{x}', \overline{y}'])$ having some intersection, and the corresponding potential new hole $[\min(\underline{x}, \underline{x}'), \min(\underline{y}, \underline{y}'), \max(\overline{x}, \overline{x}'), \max(\overline{y}, \overline{y}')]$: choose the pair for which the potential hole has the maximum area, and remove all items overlapping it. In Figure 4.2(a), the only choice is hole $[3, 3, 8, 6]$, so item 4 is removed.
3. Remove from the packing those items j that are placed at height $y_j > \alpha z(\overline{S}_{2SP})$ (α a prefixed parameter). By assuming $\alpha = 0.7$, in Figure 4.2(a), we remove items 9 and 10.
4. Shift down and left the remaining items (see above). In Figure 4.2(b), items 7 and 8 are shifted.

5. Consider the removed items by non-increasing y_j value, breaking ties by non-increasing item area: pack the current item in the lowest feasible position, left justified. In Figure 4.2(c), items 10, 9 and 4 (in this order) are re-packed.

The time complexity of the post-optimization algorithm is $O(n^2)$. Indeed, Step 1 requires $O(n^2)$ time, by considering, for each item j, all items surrounding it, in order to detect holes touching j. As there are at most $O(n)$ holes, Step 2 too requires $O(n^2)$ time. Step 3 clearly needs $O(n)$ time, while Step 4 requires $O(n^2)$ time, as previously shown. Finally, Step 5 can be implemented so as to require $O(n^2)$ time, as shown by Chazelle [24].

We have so far illustrated the core, EXPLORE, of our approach. The overall algorithm, 2SP_TS, performs a search over values of bin height H to be used within the 2BP Tabu search. At each iteration, the range $[H_1, H_2]$ of H values to be tested is halved:

algorithm 2SP_TS:
execute algorithms BUILD and TP_{2SP} of Section 4.2;
let S_{2SP} be the best solution found, and z^* its value;
compute a lower bound L by solving the 1CBP relaxation;
if $z^* = L$ **then stop**;
$H_1 := \max_j\{h_j\}$, $H_2 := z^* - 1$;
$z_1 :=$ EXPLORE(S_{2SP},H_1);
$z_2 :=$ EXPLORE(S_{2SP},H_2);
while $H_1 < H_2 - 1$ **or** a time limit has been exceeded **do**
$H := \lfloor (H_1 + H_2)/2 \rfloor$;
$z :=$ EXPLORE(S_{2SP},H);
$z^* := \min(z^*, z)$;
if $z^* = L$ **then stop**;
if $z_1 \leq z_2$ **then** $z_2 := z$, $H_2 := H$
else $z_1 := z$, $H_1 := H$
end while;
end.

4.4 Genetic algorithm

We developed a genetic algorithm, 2SP_GA, based on a permutation data structure representing the order in which the items are packed into the strip. The population size is constant and new individuals are obtained through elitist criteria and reproduction. Diversification is obtained through immigration and mutation. The research is intensified through local search. The history of the evolution is considered, in order to intensify the search in promising areas, and to escape from poor local minima.

Most of our parameters are evaluated according to the recent evolution of the algorithm. Namely, we distinguish between *improving* and *non-improving phase*: we are in a non-improving phase if the incumbent solution was not improved by the latest Q individuals generated. Computational experiments showed that a good value for distinguishing the two phases is $Q = 750$.

4.4.1 Data structure

Genotypes

The genotype of an individual is represented by a permutation $\Pi = (\pi_1, \ldots, \pi_n)$ of the items, that gives the order in which the items are packed. This data structure, adopted, for various packing problems, by several authors (see, e.g., Reeves [131], Jakobs [99], Gomez and de la Fuente [76]), has the advantage of an easy creation of new permutations and the absence of infeasible solutions. In our approach, the current sequence is given as input to algorithm TP_{2SP} of Section 4.2, that produces the

corresponding packing.

Fitness

An individual is evaluated through a fitness function $f : \Pi \to R$, computed as follows. Let v^* denote the incumbent solution value, v_0 the worse solution value in the current population, and $v(\Pi)$ the solution value produced by $\mathrm{TP}_{\mathrm{2SP}}$ for Π. The fitness of Π is then:

$$f(\Pi) = \begin{cases} \max(0,\ \beta' v^* - v(\Pi)) & \text{in an improving phase} \\ \max(0,\ \beta''(v^* + v_0)/2 - v(\Pi)) & \text{in a non-improving phase} \end{cases} \tag{4.1}$$

with $\beta', \beta'' > 1$. In this way, given two individuals, Π^a and Π^b, the percentage difference between their fitness is higher if we are in an improving phase. Hence, in an improving phase there is a high probability of selecting promising individuals, while in a non-improving phase there is a high probability of escaping from a local minimum. Good values for the parameters were experimentally established as $\beta' = 1.2$ and $\beta'' = 1.5$.

4.4.2 Evolution process

Population

We adopted a constant population size s. Initially, a first population of s random individuals is created, by ensuring that it includes no duplicated individuals. An individual is said to be *duplicated* if his item sequence can be obtained from the sequence of a different individual just by swapping identical items, i.e., distinct items having the same width and height. Sizes in the range $[50, 100]$ computationally proved good efficiency, while higher values produced a strong increase in the computing time needed by the algorithm. Hence, we adopted for s the value $s = \max(50, \min(n, 100))$.

Evolution

We adopted an elitist model (see De Jong [37]), i.e., the e individuals with highest fitness in the current generation directly pass to the new generation, with $e = s/5$ for an improving phase and $e = 1$ for a non-improving phase. An additional individual is added through intensification on the current generation (see below). The new population is then completed through immigration and generation.

The number of immigrants is $i = s/20$ for an improving phase and $i = s/10$ for a non-improving phase. Immigrants are created by ensuring that their initial triplets (π_1, π_2, π_3) are the less frequent among those of all previously generated individuals.

The remaining $s - e - i - 1$ individuals are generated through exchange of information between two parents. The parents are selected following the classical *proportional selection* approach (see Holland [91], i.e., an individual having fitness $f(\Pi^k)$ has a probability $f(\Pi^k)/\sum_{j=1}^{s} f(\Pi^j)$ of being selected. In addition, we prohibit mating between duplicated parents.

In addition, during non-improving phases, we apply mutation by randomly selecting 10% of the individuals of the new generation: for each selected chromosome, two random elements of the permutation are interchanged.

Crossover

The exchange of information between the two parents is obtained through crossover operator OX3 by Davis [36]. Let Π^a and Π^b be the selected parents. We generate two random numbers p, q $(p < q)$ in the interval $[1, n]$. The offspring is composed by two individuals, Π^c and Π^d, generated as follows. The crossover copies $\pi^a_p, \ldots, \pi^a_q$ to the same positions in Π^c, and $\pi^b_p, \ldots, \pi^b_q$ to the same positions in Π^d. Π^c and Π^d are then filled up by selecting the missing elements from Π^b and Π^a, respectively, in the same order. For example if $\Pi^a = (2, 1, 3, 7, 6, 4, 5)$, $\Pi^b = (4, 3, 1, 6, 2, 7, 5)$, $p = 3$ and $q = 4$, we obtain $\Pi^c = (4, 1, 3, 7, 6, 2, 5)$ and $\Pi^d = (2, 3, 1, 6, 7, 4, 5)$. The generated individual having the highest fitness is then selected for the new population.

We successfully compared this crossover, through computational experiments, with other crossovers proposed in the literature for similar problems, namely crossovers C1 (*Crossover One*, see Reeves [134]) PMX (*Partially Mapped Crossover*, see Goldberg and Lingle [75]), Jakobs (see [99]), OX (*Order Crossover*, see Davis [35]), UX2 (*Union Crossover 2*, see Poon and Carter [129]).

4.4.3 Intensification

We adopted local search in order to explore more deeply promising solution regions.

The following algorithm iteratively operates on a given solution S in two phases: a first attempt to re-pack the item that is most high in S, and a second attempt to rearrange the packing of items placed around the holes (see Section 4.3 for the definition of hole). The adopted approach, RE-PACK, can be outlined as follows.

1. Select as a *target* the item t that is packed most high in solution S, i.e., the one for which $y_t + h_t$ is a maximum in S.
2. Execute the following variant of algorithm $\mathrm{TP}_{2\mathrm{SP}}$ of Section 4.2. While item t is not packed, at each iteration, evaluate the position with highest touching fraction not only for the current item but also for t, and select for packing the item and the position producing the maximum touching fraction. Once item t has been packed, the execution proceeds as in $\mathrm{TP}_{2\mathrm{SP}}$. Let S' denote the resulting solution.
3. Store in S the best solution between S and S'.
4. **for each** hole in the packing pattern corresponding to S **do**
   consider the items that touch the hole's sides;
   for each pair of such items **do**
   swap the items in the permutation;
   execute algorithm $\mathrm{TP}_{2\mathrm{SP}}$
   end for
   end for
5. Let S' denote the best solution obtained at step 4.
   If $z(S') < z(S)$ store S' in S and **go to** step 1; otherwise terminate.

Recall that a different re-packing approach, still operating on holes, was used in the post-optimization phase of the Tabu search approach (see algorithm POST-OPT of Section 4.3). This approach too turned out to be useful for the genetic algorithm, and the best results were obtained by alternating the two processes as follows. Assume that the generations are numbered by integers $1, 2, \ldots$. At each odd (resp. even) generation, the individual with largest fitness not previously used for intensification (if any) is selected, and the corresponding solution S is explored through RE-PACK (resp. POST-OPT). As previously mentioned, the best individual obtained by intensification is directly added to the next generation.

4.5 A Hybrid approach

We performed extensive computational experiments both on instances from the literature and randomly generated instances (see Section 4.6). Both the Tabu search approach 2SP_TS (see Section 4.3) and the genetic algorithm 2SP_GA (see Section 4.4) were executed after the computation of upper and lower bounds through the deterministic approaches discussed in Section 4.2. The experiments showed a good empirical behavior of both metaheuristics. It turned out in particular that the genetic algorithm has a better performance for small size instances and for cases with a small value of the strip size, W, while the opposite holds for the latter. Indeed, the CPU time requested by the iterated execution of $\mathrm{TP}_{2\mathrm{SP}}$ within the genetic algorithm strongly increases with such values. In addition large values of n imply large populations, so a lesser number of them can be generated within a given time limit.

We thus implemented a third approach, based on a combination of the two algorithms. This Hybrid algorithm too starts with the computation of deterministic upper and lower bounds. Let $T_{\rm GA}$ and $T_{\rm TS}$ be prefixed time limits assigned, respectively, to 2SP_GA and 2SP_TS. Algorithm 2SP_GA is first executed, for $T_{\rm GA}$ time units, exactly as described in Section 4.4. The time check is performed at the beginning of each new generation. If the assigned time has expired, the current generation is obtained as follows. After the selection of the first $e+1$ individuals (see Section 4.4.2), the solution corresponding to the individual obtained from intensification is given as initial solution to 2SP_TS, and the Tabu search is performed for $T_{\rm TS}$ time units. When this limit has been reached, a new individual, associated with the best solution found by 2SP_TS, is added to the current population (and the incumbent solution is possibly updated by such solution). The population is then completed through i immigrants and $s-e-i-2$ generated individuals. Algorithm 2SP_GA is then executed for $T_{\rm GA}$ time units, and so on, until an overall time limit is reached.

The computational experiments showed that the performance of the hybrid algorithms is quite sensitive to values $T_{\rm GA}$ and $T_{\rm TS}$. By also considering the outcome of the experiments on the two approaches alone (see above), a reasonable definition of such values was determined as follows. Let T be the time limit assigned to each iteration (2SP_GA plus 2SP_TS) of the hybrid algorithm. Then $T_{\rm GA}=\gamma T$ and $T_{\rm TS}=(1-\gamma)T$, with

$$\gamma=\begin{cases}4/5 & \text{if } \min(n,W)\le 20\\ 1/5 & \text{if } n\ge 80 \text{ and } W>20\\ 1-n/100 & \text{otherwise}\end{cases} \tag{4.2}$$

where the third value corresponds to the segment that interpolates the two extremes.

4.6 Computational Experiments

The algorithms of the previous sections were coded in FORTRAN 77 and run on a Pentium III 800 MHz both on test instances from the literature and on randomly generated instances. All considered instances have $n\le 100$ and $W\le 500$.

Table 4.1 refers to original instances of the strip packing problem. In particular, instances `jack01`–`jack02` were proposed by Jakobs [99], instances `ht01`–`ht18` by Hopper and Turton [94], and instances `hifi01`–`hifi25` by Hifi [88]. The instances considered in Table 4.2, originally introduced for other two-dimensional cutting problems, were transformed into strip packing instances by using the item sizes and bin width.

In particular, instances `cgcut01`–`cgcut03` were proposed by Christofides and Whitlock [25], instances `beng01`–`beng07` by Bengtsson [15] and instances `ngcut01`–`ngcut12` and `gcut01`–`gcut08` by Beasley [12, 11]. Instances `jack01`–`jack02` are described in the paper by Jakobs [99], instances `hifi01`–`hifi25` are available at `ftp://panoramix.univ-paris1.fr/pub/CERMSEM/hifi/SCP`, while the ORLIB library (see Beasley [13], web site `http://www.ms.ic.ac.uk/info.html`) stores all other instances.

The initialization phase had a total time limit of 45 CPU seconds (15 seconds for the lower bound, and 30 seconds for the upper bound computation). Each metaheuristic algorithm had a time limit of 15 seconds for the lower bound computation plus 300 seconds for the exploration of the solution space (30 of which, at most, for the initial upper bound computation). The check on the elapsed CPU time was executed after each heuristic in the initialization phase, at each execution of EXPLORE in the Tabu search (and hybrid) algorithm, and at each population in the genetic (and hybrid) algorithm. As a consequence, the times in the tables can exceed the given time limit.

For each instance, Tables 4.1 and 4.2 give:

- problem name and values of n and W;
- value of the lower bound (LB);
- best upper bound value (UB) found by the heuristic algorithms of Section 4.2, with corresponding percentage gap ($\%gap$, computed as $(UB-LB)/UB$) and elapsed CPU time (T);

- percentage gap obtained by the genetic algorithm (resp. tabu search, resp. hybrid algorithm) and corresponding CPU time (*time*) elapsed when the last improvement of the incumbent solution occurred.

In Table 4.3 we give the outcome of other computational tests, performed on ten classes of randomly generated instances originally proposed in the literature for 2BP. In this case too the instances were adapted to 2SP.

The first four classes were proposed by Martello and Vigo [120], and are based on the generation of items of four different types:

type 1 : w_j uniformly random in $[\frac{2}{3}W, W]$, h_j uniformly random in $[1, \frac{1}{2}W]$;

type 2 : w_j uniformly random in $[1, \frac{1}{2}W]$, h_j uniformly random in $[\frac{2}{3}W, W]$;

type 3 : w_j uniformly random in $[\frac{1}{2}W, W]$, h_j uniformly random in $[\frac{1}{2}W, W]$;

type 4 : w_j uniformly random in $[1, \frac{1}{2}W]$, h_j uniformly random in $[1, \frac{1}{2}W]$.

Class k ($k \in \{1, 2, 3, 4\}$) is then obtained by generating an item of type k with probability 70%, and of the remaining types with probability 10% each. The strip width is always $W = 100$. The next six classes have been proposed by Berkey and Wang [16]:

Class 5 : $W = 10$, w_j and h_j uniformly random in $[1, 10]$;

Class 6 : $W = 30$, w_j and h_j uniformly random in $[1, 10]$;

Class 7 : $W = 40$, w_j and h_j uniformly random in $[1, 35]$;

Class 8 : $W = 100$, w_j and h_j uniformly random in $[1, 35]$;

Class 9 : $W = 100$, w_j and h_j uniformly random in $[1, 100]$;

Class 10 : $W = 300$, w_j and h_j uniformly random in $[1, 100]$.

For each class and value of n ($n \in \{20, 40, 60, 80, 100\}$), ten instances were generated. The time limits per instance were the same as in the previous tables. Instances and generator code are available at `http://www.or.deis.unibo.it/ORinstances/2BP/`. For each class and value of n, Table 4.3 gives:

- class and values of n and W;
- average lower bound value (LB);
- average upper bound value (UB) found by the heuristic algorithms of Section 4.2, with corresponding average percentage gap (*%gap*) and elapsed CPU time (T);
- average percentage gap obtained by the genetic algorithm (resp. tabu search, resp. hybrid algorithm) and corresponding average CPU time (*time*) elapsed when the last improvement of the incumbent solution occurred.

In addition, for each class we give the same average values over the 50 instances. The tables show a satisfactory performance of the three metaheuristic approaches. Algorithms 2SP_TS and 2SP_GA have a "complementary" behavior: 2SP_GA performs better in the case of small size instances (both for what concerns number of items and strip width), while the opposite holds for 2SP_TS. The hybrid algorithm was thus structured so as to take advantage of this phenomenon (see equation (4.2)).

The performance of the hybrid algorithm is generally (although not always) the best one: for the instances from the literature, it produced the best solution in 72 cases out of 75. Half of these instances were solved to proved optimality (i.e., by determining a solution of value equal to the lower bound). The gap obtained by the hybrid algorithm was below 5% in 62 out of 75 instances. By considering the instances not solved to proved optimality during the initialization phase, it improved the initial solution in 32 instances out of 49.

Concerning the random instances of Table 4.3, the average percentage gap of the hybrid algorithm was below 5% in 41 cases out of 50, and below 6% in 46 cases. It produced the best average percentage

Table 4.1: Results on strip packing instances from the literature. CPU seconds of a Pentium III 800 MHz.

Instance			Initialization				2SP_GA		2SP_TS		Hybrid	
Name	n	W	LB	UB	$\%gap$	T	$\%gap$	$time$	$\%gap$	$time$	$\%gap$	$time$
jack01	25	40	15	16	6.25	16.58	0.00	132.73	6.25	16.59	0.00	93.93
jack02	50	40	15	16	6.25	16.25	6.25	16.26	6.25	16.24	6.25	16.25
ht01	16	20	20	22	9.09	16.67	0.00	21.88	9.09	16.67	0.00	21.88
ht02	17	20	20	23	13.04	16.66	4.76	27.03	9.09	116.11	4.76	27.09
ht03	16	20	20	21	4.76	16.66	0.00	17.46	4.76	16.66	0.00	17.45
ht04	25	40	15	16	6.25	16.67	0.00	28.74	0.00	116.12	0.00	28.45
ht05	25	40	15	16	6.25	3.05	6.25	3.05	6.25	3.05	6.25	3.05
ht06	25	40	15	16	6.25	13.82	0.00	13.82	0.00	13.82	0.00	13.82
ht07	28	60	30	31	3.23	16.68	3.23	16.68	3.23	16.68	3.23	16.68
ht08	29	60	30	33	9.09	16.69	6.25	17.77	9.09	16.69	3.23	88.43
ht09	28	60	30	33	9.09	16.68	0.00	37.50	9.09	16.68	0.00	37.11
ht10	49	60	60	64	6.25	16.77	6.25	16.77	6.25	16.77	6.25	16.77
ht11	49	60	60	65	7.69	16.78	6.25	37.23	4.76	315.03	4.76	265.80
ht12	49	60	60	63	4.76	16.78	4.76	16.78	3.23	91.34	3.23	36.02
ht13	73	60	90	95	5.26	17.06	5.26	17.06	5.26	17.06	4.26	30.40
ht14	73	60	90	93	3.23	17.07	3.23	17.07	3.23	17.07	3.23	17.07
ht15	73	60	90	96	6.25	17.15	5.26	145.94	6.25	17.15	4.26	31.21
ht16	97	80	120	126	4.76	17.87	4.76	17.87	4.76	17.87	4.76	17.87
ht17	97	80	120	124	3.23	17.59	3.23	17.59	3.23	17.59	3.23	17.59
ht18	97	80	120	125	4.00	17.42	4.00	17.42	3.23	125.02	4.00	17.42
hifi01	10	5	13	13	0.00	0.26	0.00	0.27	0.00	0.26	0.00	0.26
hifi02	11	4	40	40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
hifi03	15	6	14	14	0.00	15.00	0.00	15.00	0.00	15.01	0.00	15.01
hifi04	11	6	19	20	5.00	15.08	5.00	15.09	5.00	15.09	5.00	15.08
hifi05	8	20	20	20	0.00	0.56	0.00	0.58	0.00	0.56	0.00	0.56
hifi06	7	30	38	38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
hifi07	8	15	14	14	0.00	0.09	0.00	0.10	0.00	0.09	0.00	0.09
hifi08	12	15	17	17	0.00	2.01	0.00	2.07	0.00	1.98	0.00	1.99
hifi09	12	27	68	68	0.00	1.66	0.00	1.66	0.00	1.66	0.00	1.67
hifi10	8	50	80	80	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00
hifi11	10	27	48	48	0.00	0.16	0.00	0.17	0.00	0.16	0.00	0.17
hifi12	18	81	34	34	0.00	0.63	0.00	0.65	0.00	0.62	0.00	0.61
hifi13	7	70	50	50	0.00	0.20	0.00	0.21	0.00	0.19	0.00	0.20
hifi14	10	100	60	70	14.29	15.17	13.04	20.76	14.29	15.17	13.04	20.75
hifi15	14	45	34	38	10.53	15.13	0.00	22.54	10.53	15.10	0.00	22.52
hifi16	14	6	32	33	3.03	15.09	3.03	15.08	3.03	15.08	3.03	15.08
hifi17	9	42	34	34	0.00	0.43	0.00	0.43	0.00	0.43	0.00	0.42
hifi18	10	70	89	101	11.88	15.09	11.88	15.11	11.88	15.10	11.88	15.10
hifi19	12	5	25	25	0.00	2.75	0.00	2.84	0.00	2.73	0.00	2.72
hifi20	10	15	19	20	5.00	15.17	5.00	15.14	5.00	15.17	5.00	15.16
hifi21	11	30	140	140	0.00	12.33	0.00	12.64	0.00	12.33	0.00	12.33
hifi22	22	90	34	38	10.53	15.08	0.00	30.78	10.53	15.08	0.00	30.57
hifi23	12	15	34	34	0.00	4.19	0.00	4.27	0.00	4.15	0.00	4.16
hifi24	10	50	103	111	7.21	15.60	6.36	30.39	7.21	15.61	6.36	30.32
hifi25	15	25	35	39	10.26	4.97	0.00	7.07	5.41	108.68	0.00	6.91

gap in 26 cases out of 50, and the best overall average percentage gap in 8 classes out of 10. With the exception of Class 9, for which most of the instances were optimally solved by the initialization phase, the hybrid algorithm improved the average gap in almost all cases (43 out of 45).

Table 4.2: Results on adapted packing instances from the literature. CPU seconds of a Pentium III 800 MHz.

Instance			Initialization				2SP_GA		2SP_TS		Hybrid	
Name	n	W	LB	UB	$\%gap$	T	$\%gap$	$time$	$\%gap$	$time$	$\%gap$	$time$
cgcut01	16	10	23	23	0.00	1.19	0.00	1.19	0.00	1.19	0.00	1.19
cgcut02	23	70	63	69	8.70	15.68	3.08	293.33	8.70	15.67	3.08	61.05
cgcut03	62	70	636	676	5.92	16.47	5.92	16.48	5.92	16.48	5.92	16.48
beng01	20	25	30	31	3.23	15.77	3.23	15.76	3.23	15.76	3.23	15.76
beng02	40	25	57	59	3.39	16.43	1.72	231.61	3.39	16.43	1.72	38.56
beng03	60	25	84	87	3.45	16.43	2.33	94.71	3.45	16.43	2.33	32.76
beng04	80	25	107	112	4.46	15.10	2.73	36.55	2.73	17.03	2.73	23.72
beng05	100	25	134	137	2.19	15.09	2.19	15.09	1.47	77.03	1.47	28.76
beng06	40	40	36	38	5.26	15.42	0.00	308.24	5.26	15.41	2.70	92.27
beng07	80	40	67	71	5.63	15.09	4.29	53.89	2.90	17.04	2.90	23.23
ngcut01	10	10	23	23	0.00	0.28	0.00	0.28	0.00	0.28	0.00	0.28
ngcut02	17	10	30	30	0.00	6.66	0.00	6.66	0.00	6.66	0.00	6.62
ngcut03	21	10	28	29	3.45	16.66	0.00	21.30	3.45	16.66	0.00	21.28
ngcut04	7	10	20	20	0.00	0.02	0.00	0.02	0.00	0.01	0.00	0.02
ngcut05	14	10	36	36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ngcut06	15	10	29	31	6.45	16.65	6.45	16.65	6.45	16.66	6.45	16.65
ngcut07	8	20	20	20	0.00	0.82	0.00	0.82	0.00	0.82	0.00	0.82
ngcut08	13	20	32	33	3.03	16.66	3.03	16.66	3.03	16.66	3.03	16.66
ngcut09	18	20	49	55	10.91	16.68	2.00	20.14	9.26	16.75	2.00	20.12
ngcut10	13	30	80	80	0.00	1.07	0.00	0.99	0.00	1.02	0.00	1.08
ngcut11	15	30	50	54	7.41	16.66	3.85	121.33	7.41	16.66	3.85	157.99
ngcut12	22	30	87	87	0.00	0.25	0.00	0.26	0.00	0.25	0.00	0.25
gcut01	10	250	1016	1016	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
gcut02	20	250	1133	1245	9.00	26.08	7.36	222.58	6.44	173.57	6.13	251.28
gcut03	30	250	1803	1803	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
gcut04	50	250	2934	3130	6.26	45.09	6.26	45.09	6.26	45.09	6.26	45.09
gcut05	10	500	1172	1284	8.72	45.77	8.72	45.77	8.72	45.77	7.93	128.29
gcut06	20	500	2514	2757	8.81	45.84	8.52	318.09	7.37	79.48	6.02	246.65
gcut07	30	500	4641	4796	3.23	51.50	3.23	51.50	3.23	51.50	2.46	267.74
gcut08	50	500	5703	6257	8.85	54.74	8.85	54.74	7.97	83.69	8.61	377.23

Table 4.3: Results for 2SSP. Random instances proposed by Martello and Vigo (Classes 1-4), and by Berkey and Wang (Classes 5-10). CPU seconds of a Pentium III 800 MHz. Average values over 10 instances.

Instance			Initialization				2SP_GA		2SP_TS		Hybrid	
Class	n	W	LB	UB	$\%gap$	T	$\%gap$	$time$	$\%gap$	$time$	$\%gap$	$time$
	20	10	60.3	61.3	1.46	10.26	1.33	11.02	1.33	37.17	1.33	10.82
	40	10	121.6	122.5	0.85	10.37	0.20	22.36	0.67	36.54	0.30	17.65
1	60	10	187.4	189.2	0.97	12.59	0.86	34.45	0.97	12.63	0.86	23.94
	80	10	262.2	263.0	0.32	7.38	0.23	7.84	0.28	13.42	0.23	8.16
	100	10	304.4	305.7	0.44	15.49	0.41	15.89	0.37	24.52	0.37	18.78
average			187.2	188.3	0.81	11.22	0.61	18.31	0.74	24.86	0.62	15.87
	20	30	19.7	21.0	6.13	16.61	1.52	23.43	4.32	26.39	0.99	41.37
	40	30	39.1	41.4	5.73	18.40	2.00	72.68	3.48	40.29	2.19	51.86
2	60	30	60.1	62.3	3.53	18.52	2.70	24.03	2.77	25.82	2.44	35.33
	80	30	83.2	85.3	2.43	17.20	2.33	17.20	1.63	64.92	1.76	29.34
	100	30	100.5	102.2	1.68	18.22	1.58	18.22	1.36	24.04	1.27	19.77
average			64.5	66.6	5.13	21.11	2.33	35.80	3.58	41.57	1.93	43.81
	20	40	157.4	167.9	6.23	20.09	4.81	79.11	5.04	64.92	4.42	94.23
	40	40	328.8	341.3	3.79	16.88	2.69	81.22	3.34	54.62	3.08	81.28
3	60	40	500.0	518.0	3.68	18.94	3.09	118.53	3.31	92.32	3.48	72.91
	80	40	701.7	721.1	2.73	19.10	2.19	106.19	2.31	71.13	2.49	50.02
	100	40	832.7	848.8	1.98	18.82	1.90	48.06	1.77	95.38	1.94	36.16
average			504.1	519.4	3.68	18.77	2.93	86.62	3.15	75.67	3.08	66.92
	20	100	61.4	70.2	12.45	18.41	6.76	66.47	8.29	35.48	6.33	78.22
	40	100	123.9	137.9	10.38	18.58	5.54	141.50	5.93	60.32	5.79	59.83
4	60	100	193.0	208.1	7.26	18.89	6.12	87.33	4.60	36.12	4.53	60.31
	80	100	267.2	284.9	6.21	19.41	6.14	37.15	4.09	46.25	4.19	27.77
	100	100	322.0	342.2	5.91	20.33	5.89	33.80	3.08	81.87	3.14	57.69
average			193.5	208.7	8.44	19.12	6.09	73.25	5.20	52.01	4.80	56.76
	20	100	512.2	549.5	7.18	19.54	4.48	103.15	5.37	67.58	4.51	72.87
	40	100	1053.8	1096.5	4.16	19.61	3.37	92.97	3.60	88.77	2.95	117.96
5	60	100	1614.0	1675.1	3.83	21.86	3.54	43.06	3.23	66.73	3.32	41.46
	80	100	2268.4	2311.5	1.88	24.93	1.81	57.51	1.57	73.17	1.73	81.70
	100	100	2617.4	2699.6	3.17	27.43	3.00	33.28	2.91	97.65	3.07	56.63
average			1613.2	1666.4	4.04	22.67	3.24	65.99	3.33	78.78	3.12	74.12
	20	300	159.9	188.0	15.00	19.98	9.15	148.47	10.46	51.68	8.56	145.96
	40	300	323.5	366.0	11.59	24.93	8.26	155.94	7.04	88.96	6.52	68.91
6	60	300	505.1	554.8	8.95	29.33	8.17	147.78	4.86	75.82	5.04	65.08
	80	300	699.7	757.3	7.59	36.83	7.44	63.27	4.43	45.11	4.43	63.00
	100	300	843.8	910.7	7.39	42.98	7.30	60.92	3.55	55.32	3.57	85.65
average			506.4	555.4	10.10	30.81	8.06	115.28	6.07	63.38	5.62	85.72
	20	100	490.4	501.9	2.45	11.19	2.45	11.21	2.45	11.21	2.45	11.21
	40	100	1049.7	1063.1	1.34	12.18	1.00	36.41	1.18	41.24	1.03	28.38
7	60	100	1515.9	1530.4	0.94	9.18	0.89	10.76	0.90	22.01	0.89	10.18
	80	100	2206.1	2230.0	1.10	13.40	0.96	44.32	1.03	41.82	0.82	72.69
	100	100	2627.0	2651.3	0.90	15.93	0.78	80.36	0.78	55.68	0.73	85.08
average			1577.8	1595.3	1.35	12.38	1.22	36.61	1.27	34.39	1.18	41.51
	20	100	434.6	480.5	9.52	19.34	7.06	194.66	8.67	43.62	7.66	128.18
	40	100	922.0	994.8	7.42	20.91	6.56	108.80	6.06	112.37	5.85	160.03
8	60	100	1360.9	1442.2	5.63	22.97	5.53	87.81	5.23	99.89	5.27	48.12
	80	100	1909.3	2019.8	5.46	26.11	5.46	26.11	4.88	85.97	5.23	57.77
	100	100	2362.8	2485.8	4.94	29.41	4.94	29.41	4.62	116.37	4.85	35.23
average			1397.9	1484.6	6.59	23.75	5.91	89.36	5.89	91.64	5.77	85.87
	20	100	1106.8	1106.8	0.00	1.11	0.00	1.11	0.00	1.11	0.00	1.11
	40	100	2189.2	2190.6	0.07	1.88	0.07	1.88	0.07	1.88	0.07	1.88
9	60	100	3410.4	3410.4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	80	100	4578.6	4588.1	0.20	6.06	0.20	6.06	0.20	6.06	0.20	6.06
	100	100	5430.5	5434.9	0.08	5.04	0.08	5.04	0.08	5.04	0.08	5.04
average			3343.1	3346.2	0.07	2.82	0.07	2.82	0.07	2.82	0.07	2.82
	20	100	337.8	359.2	6.45	18.63	4.88	82.07	5.35	65.45	4.88	99.29
	40	100	642.8	685.4	6.23	19.78	4.80	135.25	5.13	142.45	4.65	135.39
10	60	100	911.1	960.3	5.16	21.68	4.94	29.73	4.45	128.92	4.48	100.37
	80	100	1177.6	1236.1	4.70	21.91	4.69	38.80	4.23	93.30	4.61	76.07
	100	100	1476.5	1542.3	4.29	24.60	4.29	24.60	3.97	134.26	4.29	24.60
average			909.2	956.7	5.37	21.32	4.72	62.09	4.63	112.88	4.58	87.14

Chapter 5

A GA for the Two-Dimensional Knapsack Problem

[1]

In the two-dimensional knapsack problem, we are given a rectangular master surface which has to be cut into a set of smaller rectangular items, with the aim of maximizing the total value of the pieces cut. We consider the special case in which the items cannot be rotated and must be cut with their edges always parallel to the edges of the surface. We present new greedy algorithms and a hybrid genetic approach with elitist theory, immigration rate, heuristics on line and tailored crossover operators. Extensive computational results for a large number of standard and new randomly generated test problems are presented. The results show that our approach outperforms existing metaheuristics.
Key words: Knapsack Problem, Cutting Stock Problem, Genetic Algorithms.

5.1 Introduction

The two-dimensional knapsack problem (2KP) or cutting stock problem (2CSP) is known in the literature as the problem of cutting a plane rectangle into smaller rectangular items, each of a given size and value, with the objective of maximizing the total value of the items cut. This problem arises in many industrial applications; for example, the cutting of steel, glass, wood or paper and the placement of adverts in newspapers and magazines. The related problem of minimizing the amount of waste material produced by cuts can be converted into the value maximization problem by simply taking the value of each item to be proportional to its area.

The two-dimensional cutting problem can also be viewed as the problem of packing smaller items into a larger stock rectangle. Packing problems are common in the optimization of stock areas in industry layouts. In this paper, the terms *cutting* and *packing* will be used interchangeably. Many surveys are available for multidimensional cutting and packing problems. We refer to Dyckoff [48], Haessler and Sweeney [82], Sweeney and Paternoster [142], Dowsland and Dowsland [46] and Lodi, Martello and Monaci [109].

In this paper, we consider the constrained two-dimensional non-guillotine cutting problem in which each item must be cut with its edges always parallel to the edges of the master surface (i.e. orthogonal cuts) and it must have a fixed orientation (i.e. no rotation is allowed). The optimal cutting patterns are not restricted to be of the guillotine type (a guillotine cut on a rectangle runs from one edge of the rectangle to the opposite edge, parallel to the two remaining edges). Moreover, the constrained form of this problem imposes restrictions on the maximum number of items of each type (i.e. same size and value) required to be cut.

[1]The results of this chapter appears in: E. Hadjiconstantinou, M. Iori, A Genetic Algorithm for the Two Dimensional Knapsack Problem, technical report OR/02/8, *DEIS, Università di Bologna*, 2002 [81].

After the seminal work of Gilmore and Gomory [61], a number of authors have addressed the two-dimensional cutting problem in its different forms. The unconstrained non-guillotine problem has been considered by a few authors, namely, Tsai et al. *** [] [1988] *** who presented an integer programming formulation of the problem, Arenales and Morabito [4] who proposed an approach based on an AND/OR graph and a branch and bound search and Healy et al [85] who developed an algorithm based on the identification of the empty space that can be used to cut new items.

With respect to the constrained two-dimensional non-guillotine problem, optimal procedures have been presented by Beasley [12], Scheithauer and Terno [138], Hadjiconstantinou and Christofides [80], Fekete and Schepers [51, 52, 53], Amaral and Letchford [3], Boschetti et al [17] and Caprara and Monaci [22].

Beasley [12] proposed a branch and bound algorithm with an upper bound derived from a Lagrangean relaxation of a 0-1 integer linear programming formulation. Subgradient optimization was used to minimize the value of the resulting upper bound and problem reduction procedures, derived from both the original and the relaxed problems, were applied to reduce the size of the problem.

Scheithauer and Terno [138] proposed an integer programming formulation in which 0-1 variables are used to indicate whether an item is cut to the right/left or above/below another item. No computational results were given in this paper. It is interesting to note that the linear programming relaxation of this formulation was later proved to be very poor by Amaral and Letchford [3]. In the latter paper, the authors presented an upper bound which involved the solution of a large linear program by a column generation algorithm.

Hadjiconstantinou and Christofides [80] developed a branch and bound algorithm in which the search was limited using an upper bound which was based on a Lagrangean relaxation procedure and improved using subgradient optimization. Computational gains were achieved by applying new reduction tests.

Fekete and Schepers [51, 52, 53] developed a two-level tree search algorithm for solving the d-dimensional knapsack problem. In this algorithm, projections of cut items were made onto both the horizontal and vertical edges of the stock rectangle. These projections were represented by graphs in which the nodes are the cut items and an edge joins two nodes if the projections of the corresponding cut items overlap. By looking at the properties of the graphs the authors were able to check the feasibility of the corresponding patterns. In practice, the approach in [53] outperforms the previous ones.

More recenlty, Boschetti et al [17] proposed new upper bounds derived from different relaxations of a new integer programming formulation of the constrained two-dimensional non-guillotine problem. The new formulation is based on the observation that any feasible solution can be represented by two sequences in which each element is the subset of items covering the x and y positions of the master surface, respectively. The authors also presented procedures for strengthening the resulting bounds and developed new reduction tests. The computational results obtained for a number of instances drawn from the literature proved that the bounds are fast and well performing.

The most recent exact approach is due to Caprara and Monaci [22] who presented and compared four new algorithms based on the natural relaxation of 2KP which is given by the one-dimensional knapsack. In this relaxation, the knapsack has a capacity equal to the area of the master surface and the item weights are equal to their areas. Computational experimentation using a large set of instances taken from the literature shows that the proposed methods are competitive with that of [53].

Heuristic procedures for the constrained two-dimensional non-guillotine problem have been presented by Lai and Chan [102, 103], Liu and Teng [106], Leung et al [105] and Beasley [14].

Lai and Chan [102, 103] developed two heuristic algorithms based on simulated annealing and an evolutionary strategy approach, respectively. The simulated annealing algorithm consists of three steps, namely, the algorithm first divides the master surface into sub-areas that can be used for packing, then packs the items according to "as close as possible" heuristic and finally applies a classical search procedure based on moving processes and cooling schedule. The evolutionary strategy approach includes hill-climbing and mutation procedures. Both heuristic algorithms were tested on randomly generated instances as well as real world problems having the objective of minimizing the waste material.

Leung et al [105] also proposed a genetic search approach and a simulated annealing heuristic. The authors hybridize the genetic search with a simple on line *bottom-left* heuristic that packs the items as down and as left as possible. An extensive study of different crossover operators is presented but no detailed computational results are given.

The on line heuristics used in [102] and [105] cannot produce all the feasible cutting patterns. In an attempt to avoid this problem, Liu and Teng [106] used a different on line heuristic, referred to as *improved BL-algorithm*, in which, after placing the first item in the bottom left-hand corner of the master surface, all the other items were inserted starting from the top right-hand corner of the surface and then shifting them alternatingly left and down until no further shifting was possible. Unfortunately, no comparison with the works presented in [102],[103], [106] and [105] is possible since no computational results are given for problem instances drawn from the literature.

As far as we know, the state of the art in the field of heuristic procedures for the 2KP is represented by Beasley [14] who recently proposed an innovative population heuristic based on a new non linear formulation of the problem. In this formulation, 0-1 variables are used for each item to indicate whether the item is cut from the master surface or not and two other integer variables represent the coordinates of the centre of the item cut. This formulation leads to a three-dimensional encoding of a solution used to create the individuals of the population. The evolution of the population is then obtained through crossover operator and mutation. Infeasible solutions are penalized by a negative term in the fitness function and simple improvement moves are adopted in order to achieve feasibility. Computational results were presented for a number of standard problems taken from the literature as well as for a number of large randomly generated problems. Comparisons with some of the optimal algorithms considered in our literature survey above shows a good performance of the proposed approach for the smallest instances.

Metaheuristic techniques were also developed for other related two-dimensional packing problems. Regarding the two dimensional Strip Packing Problem, Jakobs [99] and Gomez and De La Fuente [76] recently developed genetic approaches. Hopper and Turton [94] presented an empirical investigation on metaheuristic and heuristic methods. Finally, Iori et al [97] presented a tabu search based on a scheme for the two-dimensional Bin Packing problem (see Lodi et al [111]) and a genetic approach using crossover operators, heuristics on line and keeping track of the development of the population. Both the approaches were improved by using intensification and local search procedures and were finally embedded into a unique algorithm.

We present here a simple but very effective heuristic approach. In a preliminary phase, initial solutions are obtained through greedy algorithms, and simple upper bounds are computed. Then a genetic search is performed. The genetic approach, defined in the following as GA_{2KP}, uses parent selections, elitist theory, immigration and different crossover operators. It is hybridized with an on line heuristic that can naturally handle the "difficult" patterns proposed in [105]. The computational results show that the algorithm finds almost always the best solution in a fast computational time, and that generally outperforms the other metaheuristics in the literature.

In Section 5.2 we briefly describe the problem and its main features. In Section 5.3 we present some simple upper bounds from the literature and some new greedy algorithms. In Section 5.4 we define our hybrid genetic approach and finally, in Section 5.5, we present extensive computational results for instances from the literature.

5.2 Problem description

The *2KP* (or either the *2CSP*) can be defined as follows: a large rectangular master surface, here defined as S, of width W_0 and length L_0 has to be cut into a number of smaller rectangular items chosen from a set of m available types. Each item type j is defined by width w_j, length l_j and value v_j, for $j = 1, ..., m$. The objective is to construct a cutting pattern for S with the highest possible total value.

The constrained *2KP* is considered, imposing that the number of replicates for each item must be lower or equal to a positive integer factor Q_j, for $j = 1, ..., m$. Some authors (e.g. [14]) address a

different version of the problem, in which the number of items to be packed is also limited inferiorly by a certain value P_j, for $j = 1, ..., m$. For simplifying our notation, in the following we denote with $M = \sum_{j=1}^{m} Q_j$ the total number of items available for the packing.

The following assumptions are made:

- S is located in the positive quadrant of the Cartesian coordinate system, with its origin (bottom left hand corner) placed in position $(0, 0)$ and with its bottom and left hand edges parallel to the x-axis and y-axis respectively.
- the cuts are restricted to be orthogonal, i.e., parallel to the x-axis or y-axis.
- All dimensions are assumed integers and therefore the cuts on the rectangles have to be made in integer steps along the x-axis or y-axis.
- The cuts are infinitely thin.
- The orientation of the items is fixed (no rotation is allowed).
- The position of an item within S is defined by the coordinates of its bottom left hand corner, referred to as the origin of the item.

Considering a fixed orientation for the items, and thus not allowing any rotation, leads to a problem with a wide range of practical applications in the cutting of materials not isomorphic, as wood and corrugated sheet. The assumption of cuts infinitely thin does not represent a loss of generality, since cuts with a finite dimension can be considered by simply modifying w_j and l_j, for $j = 1, \ldots, M$, in a preprocessing phase. Also the assumption of integer dimensions does not represent a loss of generality, since in practice real dimensions can be scaled up.

As usual in multi dimensional packing problems, the *principle of normal patterns* is adopted (see e.g. Hertz [87] and Christofides and Whitlock [25]). Formally, this principle states that a feasible pattern is equivalent to a new one in which the items are moved as down and as left as possible. When packing an item we will also refer to the concept of *corner points* as expressed in Martello and Vigo [120]. According to a tailored enumeration strategy, and given an intermediate packing, the corner points are those positions in S, in which the following items can be packed.

5.3 Upper and Lower Bounds

A simple relaxation of $2KP$ is the one given by a one dimensional knapsack (KP), in which all the items keep the same value v_j and assume a weight d_j equal to their area ($d_j = w_j l_j$) for $j = 1, \ldots, M$, while the knapsack assumes a capacity D equal to the area of S ($D = W_0 L_0$). KP can be formally stated as:

$$\max z = \sum_{j=1}^{M} v_j x_j \tag{5.1}$$

$$\sum_{j=1}^{M} d_j x_j \leq D \tag{5.2}$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, M) \tag{5.3}$$

By solving exactly this problem, we can obtain an upper bound, that we denote as U_{kp}. It has been proved that its absolute worst-case performance ratio $r_{U_{KP}}$ is quite poor. Denoting I as an instance of $2KP$, and assuming that $z(I)$ is the value of the optimal solution of I, and $U_{KP}(I)$ is the value of the solution of the KP relaxation, Caprara and Monaci [22] proved that:

$$r(U_{KP}) = \frac{1}{3} \tag{5.4}$$

The bound, although not particularly accurate, is used to evaluate the performance of the genetic algorithm, as done also in [14]. The exact solution of KP, used both for U_{kp} and for the heuristics of Section 5.3.1, is obtained in our implementation through subroutine $MT1$ by Martello and Toth [118].

5.3.1 Greedy algorithms

We implemented new heuristic algorithms, each operating following a different greedy strategy. Some of the heuristics operate by placing the items in rows forming levels, defined shelves (we recall that any feasible solution for the special case of $2KP$ in which the items must be packed in shelves is feasible for the more general case of $2KP$ here addressed). Some of the heuristics receive in input the items sorted according to a given order.

Defining $ord(j)$ for $j = 1, \ldots, M$ the index of the item in position j after sorting, we can outline our approaches as follows:

$\mathbf{HC_{KP}}$: directly derived from the algorithm proposed by Lodi et al [112] for the Bin Packing Problem, it first packs $ord(1)$ in $(0, 0)$, then completes the remaining horizontal shelf, by considering the subset $\widetilde{M}$ of items having length $l_j \leq l_{ord(1)}$, for $j \in \widetilde{M}$, and by optimally solving the KP defined by:

$$\max z = \sum_{j \in \widetilde{M}} v_j x_j \tag{5.5}$$

$$\sum_{j \in \widetilde{M}} w_j x_j \leq W_0 - w_{ord(1)} \tag{5.6}$$

$$x_j \in \{0, 1\} \quad (j \in \widetilde{M}) \tag{5.7}$$

All the items into the solution found are placed in the shelf defined by $ord(1)$. The process is then iterated by considering the next item in order not belonging to the previous shelf, and with $l_j \leq L_0 - l_{ord(1)}$, by placing it in $(0, l_{ord(1)})$ and by solving the associated shelf with a new KP. It is worth noting that KP itself is an NP-Hard problem, but it can be solved in fast time even for a large number of items, thus it does not represent a problem. The procedure ends when no more items can be packed.

$\mathbf{HC_{HV}}$: it first tries to place the items vertically one over the other, by packing $ord(1)$ in $(0, 0)$, and then by packing in $(0, h_{ord(1)})$ the first item in order that fits (if any). It iterates with the other items, until no more items can be packed vertically. Then it tries to pack them horizontally, starting by considering the first item in order not already inserted, and trying to pack it at lowest x, breaking ties by lowest y. The procedure is then iterated, alternatively vertically and horizontally, until no more items can be packed.

$\mathbf{HC_{GAP}}$: it first consider a KP on the length of S, thus maximizing the value of the items inserted with the only constraint that $\sum_{j=1}^{M} l_j \leq L_0$. The solution obtained is composed by $\widetilde{N}$ items, that form $\widetilde{N}$ associated shelves i of length l_i for $i = 1, \ldots, \widetilde{N}$. In order to complete these shelves a generalized assignment problem (GAP) is considered. By defining $\widetilde{M}$ at the set of $M - \widetilde{N}$ items not yet packed, (GAP)can be formally stated as:

$$\max z = \sum_{i=1}^{\widetilde{N}} \sum_{j \in \widetilde{M}} p'_{ij} x_{ij} \tag{5.8}$$

$$\sum_{j \in \widetilde{M}} w'_{ij} x_{ij} \leq W_0 \quad (i = 1, \ldots, \widetilde{N}) \tag{5.9}$$

$$x_{ij} \in \{0, 1\} \quad (j \in \widetilde{M}; i = 1, \ldots, \widetilde{N}) \tag{5.10}$$

in which:

$$w'_{ij} = \begin{cases} w_j & \text{if } l_j \leq l_i \\ W_0 + 1 & \text{otherwise} \end{cases} \tag{5.11}$$

$$p'_{ij} = p_j \tag{5.12}$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ inserted in shelf } i \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

for $j =\in \widetilde{M}; i = 1, \ldots, \widetilde{N}$.

The *GAP* is solved heuristically through subroutine $MTHG2$ in [118].

$\mathrm{HC}_{\mathbf{ORD}}$: it associates with each item j a corresponding shelf (obtained as in HC_{KP}) and compute the overall profit $\widetilde{p}_j$, for $j = 1, \ldots, M$, defined as the sum of the profits of the items in the shelf. The items are then sorted according to non increasing values of $\widetilde{p}_j$ and inserted in this order along the y edge, with their corresponding shelves.

$\mathrm{HC}_{\mathbf{ORD2}}$: as in HC_{ORD} it computes for each item j the corresponding shelf and the overall profits $\widetilde{p}_j$ for $j = 1, \ldots, M$. It chooses the best shelf and insert it in $(0, 0)$, and then eventually updates for each item the value of $\widetilde{p}_j$, by not considering the items already packed. The process is iterated, by inserting in S, along the y edge, the best shelf that fits, until no more shelves can be packed.

It is of interest to note that all the algorithms are called twice, interchanging the values of W_0 and L_0 and the values w_j and l_j for $j = 1, ..., M$. The solutions obtained in both cases are feasible for the original instance.

The performance of HC_{KP} and HC_{HV} changes according to the input order *ord*. To improve such performance, we tried several different orders and we kept those that computationally produced the most interesting results, mainly:

- non increasing $v_j/(w_j l_j)$
- non increasing v_j/l_j
- non increasing v_j/w_j
- non increasing $w_j l_j$
- non increasing v_j

for $j = 1, \ldots, M$.

Algorithms HC_{KP}, HC_{GAP}, HC_{ORD} and HC_{ORD2} compute solutions that are of guillotine type, and thus they can not find the best solution for all instances. The computational times required are low for all the heuristics described. Only HC_{ORD2} may need a bigger time and it is applied only to small ($M < 40$) instances. The performance is described in Section 5.5, where results are given explicitly for instances of the literature.

5.4 Genetic algorithm

The upper and lower bounds of Section 5.3 are used in conjunction with a hybrid genetic algorithm (GA_{2kp}), which is based on a permutation data structure, representing the order in which the items are given to an on line heuristic. The value of the solution obtained by the heuristic is used to evaluate the fitness of each individual. The population of individuals is of constant size (*steady state*) and it is evolved through elitist criteria, reproduction and immigration.

5.4.1 Data structure

The *coding* of an individual is represented by a permutation $\Pi = (\Pi_1, \ldots, \Pi_M)$ of the items, that gives the order in which the M items are packed by the on line heuristic. This data structure has the advantage of an easy creation of new permutations by simple recombination, and it has been used in many similar approaches for various packing problems (see, e.g., Reeves [131], Jakobs [99], Gomez and De La Fuente [76]).

The sequence (*genotype*) is then given as input to algorithm TP_{2KP}, that will described in Section 5.4.4, that produces the corresponding packing (*phenotype*) and gives the value of the heuristic solution found. After testing different *fitness* functions, we decided, according to computational evidence, to simply use the value of the solution found to represent the fitness of each individual.

Since the greedy strategy of the on line heuristic always produce feasible solutions, we do not have to deal with *infeasibility*, thus we do not use penalty functions and reconstruction algorithms common for other genetic approaches. The resulting approach is very effective since it can produce a great number of different heuristic solutions in a relatively short computational time.

5.4.2 Evolution process

The process starts by initially creating a random population of P individuals and by ensuring that it contains no duplicated individuals. An individual is said to be *duplicated* if its item sequence can be obtained from the sequence of a different individual just by swapping identical items, i.e. distinct items having the same width and length. This can happen because of the different sequence number given to the Q_j copies of item j, for $j = 1, \ldots, m$. The population is then evolved through elitist criteria, reproduction and immigration. The new individuals substitute the old ones, forming new populations of constant size P. The use of such a constant population size, normally defined as *steady state*, is common in genetic algorithms.

According to computational evidence, we adopted for P the value:

$$P = \begin{cases} 45 & \text{if } M < 15 \\ 3M & \text{if } 15 \leq M \leq 30 \\ 90 & \text{if } M > 30 \end{cases} \tag{5.14}$$

Inside the population, the individuals ℓ are sorted according to non-increasing values of their fitness f_ℓ, for $\ell = 1, \ldots, P$, computed, as said before, through the on line heuristic. According to an elitist model (see e.g. De Jong [37]), the first e individuals, forming the *elite*, directly pass to the new generation. This ensures that the new population will maintain a certain quality of solutions, but at the same time increases the probability of remaining stuck in the same local minimum. In our implementation we set $e = 1$ (i.e., only the incumbent solution belongs to the elite).

We then add to the new population c individuals obtained by *combination*. In order to generate a new individual, two parents from the previous generation are selected according to the classical *proportional selection* approach (see Holland [91], each individual has a probability of being selected directly proportional to his fitness). The information in the parents is then re-organized through crossover operators (see following Section 5.4.3), and generates an offspring of typically two individuals. The fitness of the new individuals is evaluated and only the best one of the offspring is inserted in the new population. In order to avoid combinations that have a low probability of creating solutions different from those represented by the parents, we prohibit mating among duplicated parents (duplications were eliminated in the first population but can arise in the following ones). In our implementation the number of combinations computed at each generation is set to $c = (9/10)P$.

The population is finally completed through a random creation of $r = P - e - c$ new individuals (*immigration*). This is done by ensuring that the new individuals are not duplicates of the current population. This immigration criteria has the aim of diversifying the search and of escaping from poor local minima. It proved to be a good alternative to the classical mutation approach (see e.g. [97]), which, instead, can worsen good solutions contained in the population.

In our implementation, the evolution process is stopped when a maximum computational time is elapsed, or when a maximum number of generations is reached.

5.4.3 Crossover Operators

In order to find the best way of combing together the information coming from the two parents, we tested different crossover operators, that were originally used in the literature for other packing or routing problems. Mainly:

- **C1**: one point classical crossover, used by different authors (see, e.g., Reeves [134])
- **OX**: order crossover, by Davis [35]
- **OX3**: order crossover 3, a modified version of OX by Davis [36]
- **PMX**: partially mapped crossover, developed by Goldberg and Lingle [75]
- **SJX**: originally implemented for the strip packing problem by Jakobs [99]
- **UX2**: an improved version of the original union crossover (UX), developed by Poon and Carter [129]

All the crossover operators, with the exception of UX2, generate two offspring, the best of which is then selected and inserted in the new population. UX2 generates just one single offspring, which is then directly inserted in the new population. C1 is the only operator which has a single point of interchange between the two chromosomes of the parents, while all the others use two points.

We do not give the details for all of them, since they can be found in the original papers. We focus only on OX3, because it proved to have the average best performance, and was finally used in our approach to obtain the computational results of Section 5.5. It operates as follows : let P^a and P^b be the selected parents. We generate two random numbers p, q $(p < q)$ in the interval $[1, M]$. The offspring is composed by two individuals, O^a and O^b, generated as follows. The crossover copies $P^a_p, \ldots, P^a_q$ (resp. $P^b_p, \ldots, P^b_q$) to the same positions in O^a (resp. O^b). O^a and O^b are then filled up by selecting the missing elements from P^b and P^a, respectively, in the same order. See example in Figure 5.1, where $M = 10$, $p = 5$ and $q = 6$.

P_a :	2	3	1	5	4	9	6	8	7	10
P_b :	4	1	3	2	8	9	5	10	7	6
					p		q			
O_a :	1	3	2	5	4	9	8	10	7	6
O_b :	2	3	1	5	8	9	4	6	7	10

Figure 5.1: Crossover operator O3.

It is important to note that, considering that all the items appear just once in the offspring and considering the greedy strategy of the heuristic on line, all the solutions resulting from the combination phase are feasible. GA_{2KP} showed good performance with O3, PMX, SJX and UX2, while the value of the solutions found tended to worsen with C1 and OX.

5.4.4 On line heuristic

The performance of the hybrid genetic algorithm is strictly related to the behavior of the heuristic used at each iteration, both in terms of time and of quality of the solution found. Different heuristics have been implemented and tested within our approach for achieving the best results. Finally, procedure TP_{2KP}, based on the evaluation of the *touching perimeter*, in the following defined as TP, among the items and the surface S, was adopted.

Similar approaches to TP_{2KP} proved to have good performance for other related 2-dimensional packing problems, as in Lodi et al [112] for a particular case of the Bin Packing problem, or in Iori et al [97] for the Strip Packing problem.

TP_{2KP} receives in input the chromosome $\Pi = (\Pi_1, \ldots, \Pi_M)$. It packs Π_1 in $x = 0$, $y = 0$ and then computes the sets of *extreme positions* $\widetilde{X}$ and $\widetilde{Y}$, along x and y edges, representing the coordinates in which the following items can be placed. After placing O_1, we have that $\widetilde{X} = \{0, w_{(O_1)}\}$ and $\widetilde{Y} = \{0, l_{(O_1)}\}$. Item O_2 is then considered for the packing. For each $x \in \widetilde{X}$ and $y \in \widetilde{Y}$ for which the packing would be feasible, we consider the touching perimeter (TP), i.e. the fraction of the perimeter of Π_2 that would touch either the edges of Π_1, either the edges of the surface S. If no feasible position exists, the procedure does not pack Π_2 and iterates with the next item, otherwise, the position for which the touching perimeter is the highest is finally chosen for packing Π_2. The procedure goes on iteratively, until no more items can be packed and returns the value of the solution found z and the corresponding pattern. The algorithm is represented in Figure 5.2.

Procedure TP$_{2KP}$
1. pack item Π_1 in (0,0);
2. set $\widetilde{X} = \{0, w_{(O_1)}\}$, $\widetilde{Y} = \{0, l_{(O_1)}\}$, $z = v_{(O_1)}$
3. **for** each item $j = \Pi_2, \ldots, \Pi_M$ **do**
   - 3.1 **for** each $x \in \widetilde{X}$ in order **do**
     - **for** each $y \in \widetilde{Y}$ in order **do**
       - **if** insertion of j in (x, y) is feasible **then**
         - compute TP;
         - **break** cycle y;
   - 3.2 **if** a feasible position exists **then**
     - pack j in position with highest TP;
     - set $z = z + v_j$;
     - insert in order new positions x (resp. y), if any, in $\widetilde{X}$ and $\widetilde{Y}$;

4. **return**(z);

Figure 5.2: On line heuristic called iteratevely by GA_{2kp}

The behavior of the algorithm is explained in Figure 5.3, in which, after having packed items 1, 2, 3 and 4, we want to pack item 5. The subsets of extreme positions are $\widetilde{X} = \{0, 7, 11, 18, 23\}$ and $\widetilde{Y} = \{0, 3, 4, 7, 10\}$. We first try position a, obtaining $TP_a = 19$, then b, which is infeasible, and c, with $TP_c = 6$. Whenever we find, for a certain (x, y), a feasible cutting of the new item, the following positions (x', y'), with $x' = x$ and $y' > y$ are not explored, since not particularly relevant. Positions d and e have respectively $TP_d = 7$ and $TP_d = 14$, while f is again not feasible. Position a is finally chosen for the packing of item 5.

When two or more positions have equal values of TP, ties are broken randomly. This proved to be very effective, since it can allow TP_{2KP} to achieve difficult patterns, and can give a greater variability to the overall approach. During the genetic search, TP_{2KP} is called interchanging values of w_j with l_j, for $j = 1, \ldots, M$ and W_0 with L_0, as done for the greedy heuristics. The best solution found among the two is kept.

We can notice that more chromosomes can represent the same patterns. It is important to extract from each chromosome the best information (i.e. the best resulting pattern). For this, we disregarded

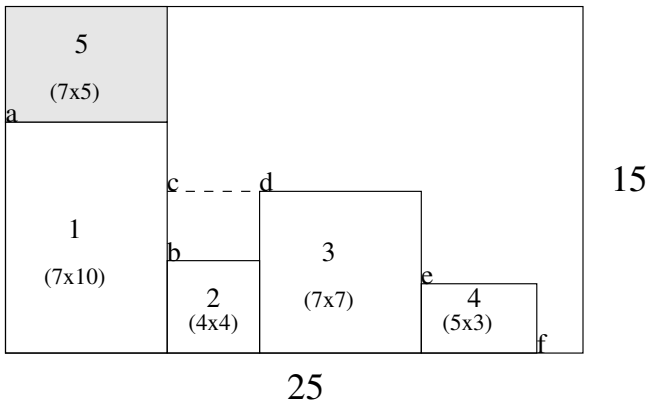

Figure 5.3: Algorithm TP_{2KP}

the use of *corner points* (see e.g. Martello, Pisinger and Vigo [116]). In Figure 5.3, indeed, position b does not represent a corner point, but we consider it in any case. The use of this tool has proved very convenient for the reduction of enumeration trees, but it is not helpful in the use of on line heuristics.

Other heuristics that were used in similar approaches, as in [102], [103] and [105], are based on the idea of inserting the next item in the position which is as down and as left as possible (*bottom left, BL*) or on similar ideas. As it has been noted in [106], some patterns (see figure 5.4) cannot be obtained by the BL algorithm.

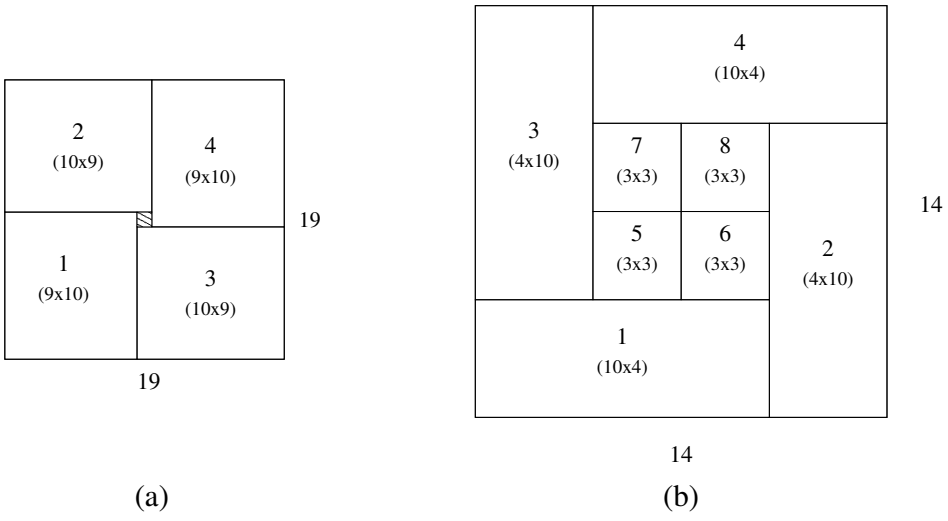

Figure 5.4: Impossible patterns for BL heuristic

It is interesting to note that our algorithm can easily produce patterns in Figure 5.4 (a), when receiving in input orders as $O = (1, 3, 2, 4)$ or similar. It can also reproduce patterns in Figure 5.4 (b), when receiving orders as $O = (1, 3, 2, 5, 6, 7, 8, 4)$ or similar.

5.5 Computational Experiments

The algorithms of the previous sections were coded in FORTRAN 90 and run on a Pentium IV 1700 MHz, 128 MB RAM, with Windows 2000 operating system, on test instances from the literature.

Table 5.1 refers to original instances of different versions of $2KP$. In particular, instances `cgcut01`–`cgcut03` were proposed by Christofides and Whitlock [25], instances `ngcut01`–`ngcut12` and `gcut01`–`gcut13` by Beasley [12, 11], instances `okp01`–`okp05` by Fekete and Schepers [52], instances `hccut01`–`hccut05` by Hadiconstantinou and Christofides [80], and instance `Wang20` by Wang [147]. Instances `gcut01`–`gcut08` and `cgcut01`–`cgcut03` were originally created respectively for guillotine $2KP$ and constrained guillotine $2KP$, but can be considered with non-guillotine cuts (see e.g. [22]).

Table 5.2 refers to instances `ngcutfs01`, `ngcutfs02` and `ngcutfs03`, recently created by Beasley [14], according to the generation criteria of Fekete and Schepers [52]. The majority of the instances is available in the ORLIB library (see Beasley [13]), at the web site `http://www.ms.ic.ac.uk/info.html`.

Regarding Table 5.1, for each instance, m represents the number of different items and M the total number of items, while W_0 and L_0 give the width and the length of the surface S, as defined in Section 5.1. U_{kp} represents the value of the upper bound computed as in Section 5.3, while Z_0 is the best solution found by the greedy heuristics of Section 5.3.1. The best solution found by the overall approach is reported in column Z, (where the bold character is used when Z is equal to the value of the solution proved to be optimal), while $\%_{gap}$ represents the percentage gap of Z from the optimal solution or from the bound, computed as $100(Z - \widetilde{Z})/\widetilde{Z}$, where $\widetilde{Z}$ is equal to the exact solution, if known, to the value of U_{kp} otherwise. Finally, the number of generations performed by the genetic algorithm and the CPU time elapsed in order to find Z are represented in column n_{gen} and $Time$. For each group of instances the average values of $\%_{gap}$, n_{gen} and $Time$ are given, while the last row represents the overall average on the whole set of instances.

The performance of the algorithm, as usual in the metaheuristic field, can change slightly since the algorithm is randomized. The values reported in Table 5.1 represent the best values found over 10 different tries.

For those instances for which the optimal solution is known, we decided to stop the genetic search whenever this value was found (as done e.g. in [14]). For all instances, we set the maximum number of generations equal to 3000, and the maximum time limit equal to 1200 CPU seconds.

The performance of GA_{2KP} is very good, since in reasonable computational times it can achieve the best solution for almost all the instances. Only for `gcut12` we were not able to reach the optimum, while for `gcut13` we managed to find a new best solution (the optimal value is still unknown).

In Table 5.2, column m represents the number of items, column Q the maximum replication for each item, and column M the resulting total number of items. For each M, 10 instances are available. Type I, II and III differ from the creation of the values: in type I the ratio between v_j and $w_j l_j$ is equal to 1, in type II it is equal to 2 and in type III to 3. Columns $\%_{gapH}$ gives the average (on the ten instances) percentage gap for the first set of greedy heuristics, while $\%_{gap}$ gives the average percentage gap of the overall algorithm (both computed as in Table 5.1). Column n_{gen} and column $Time$ give the percentage number of generations computed and the time elapsed, when the best heuristic solution was found. For these instances, results refer to just one try of GA_{2KP}. The maximum number of generations for the algorithm was set to 1500, and the maximum time to 600 CPU seconds.

In Table 5.3 and in Table 5.4, we compare our results with the most relevant algorithms known in the literature for this problem. Algorithm $B03$ is the population heuristic developed by Beasley in [14], $FS97$ is the exact algorithm by Fekete and Schepers [53], and $CM03$ is the best exact approach among those proposed by Caprara and Monaci (mainly, algorithm $A3$ in [22]). The computational times here reported are obtained from the original papers, and compared with our PC using Dongarra working paper [42], considering that Beasley used a Silicon Graphics O2 workstation (with R10000 chip, 225 MHz, 128 MB main memory, Fekete and Schepers a Sun workstation with Ultra SPARC processors, 175 MHz, and Caprara and Monaci a Pentium III, 800 MHz. In Table 5.3, we report the times for the achievement of the best solution for the instances of the literature, where $*$ means that the solution found is optimal.

It is of course unfair to compare a metaheuristic with exact algorithms. For small instances, the

CPU times required for achieving the optimal solutions are generally low, both for the exact (with proof of optimality) then for GA_{2KP}. However, our algorithm shows a very good performance and can achieve almost always the optimum in similar computational times as the exact. It also has a general better performance than $B03$. For the 21 instances in which the performance of $B03$ is given in explicit, GA_{2KP} outperforms it 8 times, and it is never outperformed, with generally lower times.

In Table 5.4 we again compare our results with $B03$, for the bigger instances `ngcutfs01`, `ngcutfs02` and `ngcutfs03`,for which no exact solution is known. Also in these case our approach generally outperform $B03$, obtaining lower gaps for 17 averages over 21. Also the CPU times are good, since the overall averages for GA_{2KP} are always smaller than those of $B03$.

As a conclusion, we can say that GA_{2KP} manages to achieve good solutions in a generally low CPU time, and low number of generations. The solutions are almost always optimal when the optimum is known, and within small gaps from U_{kp} bound, for those instances where no optimal solution is known.

Table 5.1: Results on instances from the literature. CPU seconds on a Pentium IV 1700 MHz.

Instance	**m**	**M**	$\mathbf{W_0}$	$\mathbf{L_0}$	$\mathbf{UB_{1kp}}$	$\mathbf{Z_{start}}$	**Z**	$\%_{gap}$	$\mathbf{n_{gen}}$	**Time**
ngcut01	5	10	10	10	201	156	**164**	0.00	0	0.000
ngcut02	7	17	10	10	253	230	**230**	0.00	0	0.000
ngcut03	10	21	10	10	266	238	**247**	0.00	8	0.031
ngcut04	5	7	15	10	275	268	**268**	0.00	0	0.000
ngcut05	7	14	15	10	373	358	**358**	0.00	0	0.000
ngcut06	10	15	15	10	317	267	**289**	0.00	6	0.008
ngcut07	5	8	20	20	430	267	**430**	0.00	0	0.000
ngcut08	7	13	20	20	938	753	**834**	0.00	1	0.000
ngcut09	10	18	20	20	962	781	**924**	0.00	1	0.000
ngcut10	5	13	30	30	1517	1452	**1452**	0.00	0	0.000
ngcut11	7	15	30	30	1864	1434	**1688**	0.00	1	0.000
ngcut12	10	22	30	30	2012	1721	**1865**	0.00	3	0.031
Average								0.00	1.7	0.006
hccut03	7	7	30	30	1347	1178	**1178**	0.00	0	0.000
hccut08	15	15	30	30	1547	1270	**1270**	0.00	0	0.000
hccut11	10	10	40	40	3079	2517	**2517**	0.00	0	0.000
hccut12	15	15	40	40	3604	2921	**2949**	0.00	0	0.016
Average								0.00	0	0.003
okp01	15	50	100	100	29133	24533	**27718**	0.00	238	26.695
okp02	30	30	100	100	24800	20339	**22502**	0.00	16	1.656
okp03	30	30	100	100	26714	23282	**24019**	0.00	18	1.797
okp04	33	61	100	100	33631	30529	**32893**	0.00	12	1.383
okp05	29	97	100	100	29045	25014	**27923**	0.00	832	117.930
Average								0.00	223.2	29.892
cgcut01	7	16	15	10	260	244	**244**	0.00	0	0.000
cgcut02	10	23	40	70	2919	2355	**2892**	0.00	14	0.461
cgcut03	20	62	40	70	2020	1520	**1860**	0.00	1153	36.914
Average								0.00	389	12.458
wang20	20	42	70	40	2800	2277	**2726**	0.00	0	0.051
gcut01	10	10	250	250	62488	48368	**48368**	0.00	0	0.000
gcut02	20	20	250	250	62500	46778	**59798**	0.00	200	120.590
gcut03	30	30	250	250	62500	52960	**61275**	0.00	9	7.810
gcut04	50	50	250	250	62500	59813	**61380**	0.00	27	23.910
gcut05	10	10	500	500	245263	177207	**195582**	0.00	0	0.970
gcut06	20	20	500	500	248007	224166	**236305**	0.00	4	12.930
gcut07	30	30	500	500	247663	194229	**240143**	0.00	4	18.760
gcut08	50	50	500	500	249191	207116	**245758**	0.00	127	669.370
gcut09	10	10	1000	1000	1000000	822004	**939600**	0.00	5	55.780
gcut10	20	20	1000	1000	1000000	877079	**937349**	0.00	12	155.910
gcut11	30	30	1000	1000	1000000	817617	**969709**	0.00	19	434.940
gcut12	50	50	1000	1000	1000000	918175	*976877	0.27	1	12.870
gcut13	32	32	3000	3000	9000000	8382936	**8408316	6.57	0	4.535
Average								0.53	31.4	116.798
Overall								0.01	69.5	43.727

(*) optimal solution=979521
(**) best solution known

Table 5.2: Results on instances `ngcutfs01`, `ngcutfs02` and `ngcutfs03`. CPU seconds on a Pentium IV 1700 MHz.

m	**Q**	**M**	**Type I**				**Type II**				**Type III**			
			$\%_{gapH}$	$\%_{gap}$	n_{gen}	*Time*	$\%_{gapH}$	$\%_{gap}$	n_{gen}	*Time*	$\%_{gapH}$	$\%_{gap}$	n_{gen}	*Time*
40	1	40	12.390	5.346	20.8	2.055	17.774	6.913	202.8	23.221	22.993	6.111	367.5	40.445
	3	120	8.936	3.358	335.6	43.446	7.522	2.934	183.1	25.091	9.575	2.169	497.9	76.547
	4	160	6.664	2.200	256.7	33.646	9.207	2.772	100.5	15.736	13.030	2.217	236.7	46.848
50	1	50	11.057	4.268	63.3	6.587	13.452	4.845	235.1	25.775	14.200	4.575	306.9	33.829
	3	150	5.048	2.199	568.4	76.798	5.144	1.656	53.0	8.282	8.883	1.820	385.7	72.577
	4	200	7.594	1.342	198.1	31.007	7.049	2.114	461.4	74.914	8.827	2.130	288.2	51.464
100	1	100	3.920	1.602	269.5	29.884	7.087	1.558	187.6	24.890	8.676	1.901	88.2	13.325
	3	300	3.666	0.966	453.2	85.998	4.795	0.870	567.2	121.789	5.037	1.140	367.8	91.737
	4	400	3.650	0.997	295.5	73.348	4.626	0.905	250.9	65.090	6.087	0.662	385.5	129.025
150	1	150	4.421	0.859	359.2	49.238	4.408	0.863	444.7	62.986	5.829	1.453	316.5	50.129
	3	450	1.977	0.323	119.4	37.311	2.195	0.335	231.6	65.777	2.868	0.316	455.5	152.927
	4	600	1.725	0.602	276.4	125.376	2.147	0.567	352.0	148.565	3.358	0.618	306.9	143.597
250	1	250	2.496	0.739	325.5	54.904	3.177	0.762	284.3	57.488	3.813	0.735	315.7	66.574
	3	750	1.801	0.460	500.5	212.058	1.673	0.306	222.5	113.914	2.866	0.756	197.4	114.210
	4	1000	1.334	0.138	81.4	108.324	1.242	0.378	173.9	130.616	2.760	0.332	264.8	237.479
500	1	500	1.201	0.170	290.5	100.690	1.889	0.246	420.1	137.084	2.859	0.206	408.5	149.623
	3	1500	1.397	0.175	161.5	202.890	1.114	0.113	93.1	137.608	1.524	0.282	89.8	142.671
	4	2000	0.921	0.146	118.0	211.154	0.786	0.171	61.3	124.561	1.500	0.246	129.0	304.746
1000	1	1000	0.527	0.065	147.0	99.045	1.039	0.205	133.8	108.928	1.609	0.186	51.0	49.265
	3	3000	0.655	0.045	50.9	188.874	0.622	0.081	67.6	267.941	1.131	0.221	85.0	392.061
	4	4000	0.527	0.137	28.6	176.468	0.878	0.080	40.8	283.346	0.933	0.287	44.9	320.377
Average			3.900	1.244	234.2	92.814	4.658	1.365	227.0	96.362	6.112	1.350	266.1	127.593
Overall Average							4.890	1.320	242.4	105.589				

Table 5.3: Comparisons with other algorithms from the literature.

	Metaheuristics				Exacts			
Instance	**GA**		**B03**		**FS97**		**CM03**	
ngcut01	0.000	*	0.003	*	0.003	*	-	
ngcut02	0.000	*	0.027	*	0.004	*	-	
ngcut03	0.031	*	0.088	*	0.008	*	-	
ngcut04	0.000	*	0.002	*	0.001	*	-	
ngcut05	0.000	*	0.018	*	0.001	*	-	
ngcut06	0.008	*	0.072	*	0.012	*	-	
ngcut07	0.000	*	0.002	*	0.001	*	-	
ngcut08	0.000	*	0.542	*	0.008	*	-	
ngcut09	0.000	*	0.363	*	0.003	*	-	
ngcut10	0.000	*	0.005	*	0.002	*	-	
ngcut11	0.000	*	0.100	*	0.012	*	-	
ngcut12	0.031	*	0.580		0.024	*	-	
hccut03	0.000	*	0.005	*	0.001	*	-	
hccut08	0.000	*	-		0.002	*	-	
hccut11	0.000	*	0.007	*	0.018	*	-	
hccut12	0.016	*	-		0.002	*	-	
okp01	26.695	*	3.285		1.055	*	4.088	*
okp02	1.656	*	2.198		10.567	*	257.482	*
okp03	1.797	*	1.910		6.639	*	0.287	*
okp04	1.383	*	5.347		4.554	*	0.247	*
okp05	117.930	*	13.907		3.649	*	81.277	*
cgcut01	0.000	*	-		-		0.243	*
cgcut02	0.461	*	-		-		88.835	*
cgcut03	36.914	*	1.438		0.679	*	0.745	*
wang20	0.051	*	1.143		1.137	*	0.453	*
gcut01	0.000	*	-		-		0.000	*
gcut02	120.590	*	-		-		0.023	*
gcut03	7.810	*	-		-		0.205	*
gcut04	23.910	*	-		-		54.683	*
gcut05	0.970	*	-		-		0.000	*
gcut06	12.930	*	-		-		0.000	*
gcut07	18.760	*	-		-		0.045	*
gcut08	669.370	*	-		-		19.562	*
gcut09	55.780	*	-		-		0.000	*
gcut10	155.910	*	-		-		0.000	*
gcut11	434.940	*	-		-		0.773	*
gcut12	12.870		-		-		2.243	*
gcut13	4.535		-		-		>300.000	

Table 5.4: Comparisons for instances *ngcutfs*.

Instance	**Type I**				**Type II**				**Type III**			
	GA		**B03**		**GA**		**B03**		**GA**		**B03**	
	%g	**T**	**%g**	**T**	**%g**	**T**	**%g**	**T**	**%g**	**T**	**%g**	**T**
40	3.635	26.382	4.691	4.489	4.206	21.349	4.940	6.056	3.499	54.613	4.767	10.172
50	2.603	38.131	3.470	6.050	2.872	36.324	3.620	7.472	2.842	52.623	3.363	12.078
100	1.188	63.077	1.442	11.806	1.111	70.590	1.640	16.233	1.234	78.029	1.509	25.572
150	0.595	70.642	0.925	23.311	0.588	92.443	0.873	28.494	0.796	115.551	1.028	40.689
250	0.446	125.095	0.654	50.589	0.482	100.673	0.560	63.994	0.608	139.421	0.625	87.056
500	0.164	171.578	0.209	143.711	0.177	133.084	0.180	176.044	0.245	199.013	0.234	214.367
1000	0.082	154.796	0.065	411.178	0.122	220.072	0.068	481.517	0.231	253.901	0.091	578.350
Average	1.244	92.814	1.637	93.017	1.365	96.362	1.697	111.400	1.35	127.593	1.660	138.333

Part IV

Algorithms for Routing Problems

Chapter 6

On the CVRP with Two-Dimensional Constraints

[1]

We consider a special case of the symmetric capacitated vehicle routing problem, in which a fleet of K identical vehicles must serve n customers, each with a given demand consisting in a set of rectangular two-dimensional weighted items. The vehicles have a two-dimensional loading surface and a maximum weight capacity. The aim is to find a partition of the customers into routes of minimum total cost and such that, for each vehicle, the weight capacity is taken into account and a feasible two-dimensional allocation of the items into the loading surface exists.

The problem has several practical applications in freight transportation and it is $\mathcal{NP}$-Hard in the strong sense. We propose an exact approach, based on a branch-and-cut algorithm, for the minimization of the routing cost, that iteratively calls a branch-and-bound algorithm for checking the feasibility of the loadings. Heuristics are also used in order to improve the overall performance of the algorithm. The effectiveness of the approach is shown by means of computational results.

6.1 Introduction

Given a central depot, the Vehicle Routing Problem (VRP) calls for the determination of the optimal set of routes to be performed by a fleet of vehicles, in order to satisfy the demand of a given set of customers. Several important variants of this basic problem were extensively studied in the literature (see, e.g., Toth and Vigo (2002b) for a recent review). In particular, the symmetrical Capacitated VRP (CVRP) is the well-known variant of the VRP where all vehicles are identical and have a maximum loading capacity, and all the arcs in the graph representing the underlying road network can be travelled along both directions, producing the same cost.

In the CVRP literature, the demand of each customer is generally expressed by a positive integer that represents the total weight or volume of the demanded items. In this case, checking the feasibility of a solution simply requires to ensure that the sum of the demands of the customers assigned to each vehicle does not exceed its total loading capacity. However, in many practical freight distribution applications, the loading of the items into the vehicles can represent a difficult problem, especially for large-size items. In this case, the loading pattern of the items on each vehicle must be found in order to achieve feasible solutions to the routing problem. These loading issues may have a great impact on the sequencing of the customers along the routes, and common examples of such situations may be found in the distribution of goods as furniture, mechanical components and household appliances.

[1] The results of this chapter appears in: M. Iori, J.J. Salazar Gonzalez, and D. Vigo, An exact approach for the symmetric capacitated vehicle routing problem with two dimensional loading constraints, technical report OR/03/04, DEIS, Università di Bologna, 2003 [98].

Many of these applications incorporate relevant additional features that influence the actual loading problem to be solved. For example, the items are often transported on top of rectangular bases, e.g., large pallets of suitable size, that may neither be stacked one over the other, nor rotated during the loading and unloading process. In this case, the general three-dimensional loading problem reduces to a suitably defined two-dimensional loading problem of the rectangular item bases on the vehicle loading surface. In addition, when the demand of a customer is made up by more than one item, all such items should be assigned to the same vehicle in order to avoid split deliveries. Finally, vehicles are generally rear-loaded and load rearrangement at the customer sites can be difficult, time-consuming or even impossible, due to the weight, size, and fragility of the items.

In this paper we investigate a variant of the CVRP with the above-described specific two-dimensional loading additional constraints, hereafter denoted as 2L-CVRP. In the 2L-CVRP all vehicles are identical, have a known weight capacity and a single rectangular loading surface that may be accessed only from one side. The demand of each customer is defined by a set of rectangular items with given size and weight. All the items of a given customer must be assigned to a single vehicle. We suppose that no load re-arrangement is possible at customer sites. Thus, when unloading items of one customer, no item of a customer to be visited later along the route can be moved. These requirements will be referred to, in the following, as the *item clustering* and the *sequential loading* constraints, respectively.

The 2L-CVRP combines the classical and extensively studied CVRP, with a loading problem that is closely related to the well-known Two-Dimensional Bin Packing Problem (2BPP). The 2BPP calls for the determination of a packing pattern of a given set of rectangular items into the minimum number of identical rectangular bins.

It is clear that the 2L-CVRP is strongly $\mathcal{NP}$-Hard since it generalizes the CVRP. Moreover, the two problems which are combined to obtain the 2L-CVRP are extremely difficult to be solved. State-of-the-art exact approaches for the CVRP have solved instances with up to 135 customers, but within a reasonable computing time (i.e., some hours on a common PC) they are not likely to solve instances with more than 50-60 customers (see, e.g., Toth and Vigo (2002b)).

Also the 2BPP is very difficult to be solved in practice. Exact algorithms and lower bounds were recently proposed by Martello and Vigo (1998), Fekete and Schepers (2003), Boschetti and Mingozzi (2003a, 2003b) and Pisinger and Sigurd (2003). Exact approaches for the 2BPP are generally based on branch-and-bound techniques and are able to solve instances with up to 100 items, but fail in many cases for smaller instances (indeed, some instances with just 20 items still remain unsolved). In the literature, the problems derived from the 2BPP with some additional side constraints are generally included in the category of the Container Loading Problems (2CLP). For the 2CLP, heuristic approaches have been proposed by Pisinger (1998, 2002) and Bortfeld and Gehring (2000), an analytical model was proposed by Chen et al. (1995) and an LP-based bound was presented by Scheithauer (1992, 1999).

The 2L-CVRP was not previously studied in the literature. The only closely-related reference we are aware of is on a VRP with three-dimensional loading constraint introduced by Türkay (2003), who proposed a general integer programming model derived from the Container Loading model proposed by Chen et al. (1995). The resulting approach was used to solve an instance involving 5 items and 5 customers. In addition, pickup and delivery problems with rear-loaded vehicles were also studied in the literature, see, e.g., Xu et al. (2003).

This paper presents an exact algorithm for the solution of the 2L-CVRP based on a branch-and-cut approach, that is currently the most successful technique for the solution of related problems as the CVRP (see, e.g., Naddef and Rinaldi (2002)). Within the algorithm we used both basic classes of valid inequalities derived for the CVRP, as well as specific valid inequalities associated with infeasible loading sequences. To this end, the feasibility of a given loading pattern is determined through lower bounds, effective heuristics and a specialized branch-and-bound algorithm. The overall algorithm was extensively tested on benchmark instances derived from the CVRP test problems and showed a satisfactory behavior, being able to optimally solve instances with up to 35 customers and more than 100 items.

The paper is organized as follows. Section 6.2 describes in details the new problem and introduces the necessary notation. Section 6.3 presents an exact approach derived from the standard branch-

and-cut algorithm for the CVRP, strengthened by new inequalities that take into account the vehicle loading component of the 2L-CVRP. The separation of these loading constraints is addressed in Section 6.4 through a specific branch-and-bound procedure. Section 6.5 examines the computational results and Section 6.6 draws some conclusions.

6.2 Problem Description

The 2L-CVRP may be defined as the following graph-theoretical problem. We are given a complete undirected graph $G = (V, E)$, in which V defines the set of $n + 1$ vertices corresponding to the depot (vertex 0) and to the customers (vertices $1, \ldots, n$). For each edge $e \in E$ the associated travelling cost, c_e, is defined. In the following, a given edge e can be also represented by its endpoint vertices $\{ij\}$.

A set of K identical vehicles is available at the depot. Each vehicle has a weight capacity D and a rectangular loading surface, that is accessible from a single side for loading and unloading operations, whose width and height are equal to W and H, respectively. We also denote by $A = WH$ the total area of the loading surface.

Each customer i ($i = 1, \ldots, n$) is associated with a set of m_i rectangular items, whose total weight is equal to d_i, and each having specific width and height equal to $w_{i\ell}$ and $h_{i\ell}$, ($\ell = 1, \ldots, m_i$), respectively. Each item will be denoted by a pair of indices (i, ℓ). In addition, we denote by $a_i = \sum_{\ell=1}^{m_i} w_{i\ell} h_{i\ell}$ the total area of the items of customer i. Finally, let $M = \sum_{i=1}^{n} m_i$ denote the total number of items. Without loss of generality, we assume that all the above input data are positive integers.

The loading of the items on a vehicle is subject to the following specific constraints:

Item clustering: All the items of a given customer must be loaded on the same vehicle. Therefore, for each customer i, we assume that $d_i \leq D$ and there exists a feasible two-dimensional loading of the m_i items into a single vehicle surface ($i = 1, \ldots, n$).

Item orientation: Items have a fixed orientation (i.e., they may not be rotated) and must be loaded with their sides parallel to the sides of the loading surface (i.e., a so-called *orthogonal loading* is required).

Sequential loading: Unloading operations at the customers' site are performed through a single side of the vehicle and may not result in a load rearrangement on the vehicle or in lateral shifts of the item to be unloaded. Therefore, when unloading the items of a customer, no item belonging to customers served later along the route may either be moved or lay in front of the item to be unloaded.

We now give a more formal definition of a *feasible route* in our routing problem. A feasible route is associated with a subset S of customers and with a sorting denoted by a bijection $\sigma : S \to \{1, \ldots, |S|\}$, where $\sigma(i)$ is the position in which customer $i \in S$ is visited along the route. A first condition that a route (S, σ) must satisfy is relative to the item weights, and is the classical capacity restriction of the CVRP:

Condition 1: The weight capacity of the vehicle may not be violated, i.e.,

$$\sum_{i \in S} d_i \leq D$$

The remaining conditions refer to the loading pattern. It is convenient to map the loading surface with the positive quadrant of a Cartesian coordinate system, whose origin $(0, 0)$ corresponds to the bottom-left corner of the loading surface in the vehicle, and where the x-axis and y-axis correspond to its bottom and left sides, respectively. The position of an item (i, ℓ) within the vehicle can be defined by two variables $x_{i\ell}$ and $y_{i\ell}$, representing the coordinates of its bottom-left corner in the loading surface. The back of the vehicle, from where items can be loaded and unloaded, is placed at height H. Then, the values for these variables must satisfy the following conditions:

Condition 2: The items must be completely contained within the loading surface, i.e.,

$$0 \leq x_{i\ell} \leq W - w_{i\ell} \qquad \text{and} \qquad 0 \leq y_{i\ell} \leq H - h_{i\ell}$$

for all $i \in S$ and $\ell \in \{1, \ldots, m_i\}$.

Condition 3: No two items can overlap, i.e.,

$$x_{i\ell} + w_{i\ell} \leq x_{j\ell'} \quad \text{or} \quad x_{j\ell'} + w_{j\ell'} \leq x_{i\ell} \quad \text{or} \quad y_{i\ell} + h_{i\ell} \leq y_{j\ell'} \quad \text{or} \quad y_{j\ell'} + h_{j\ell'} \leq y_{i\ell}$$

for all $i, j \in S$, $\ell \in \{1, \ldots, m_i\}$, $\ell' \in \{1, \ldots, m_j\}$, and $(i, \ell) \neq (j, \ell')$.

Condition 4: This condition arises from the sequential loading requirement, i.e.,

$$y_{i\ell} \geq y_{j\ell'} + h_{j\ell'} \quad \text{or} \quad x_{i\ell} + w_{i\ell} \leq x_{j\ell'} \quad \text{or} \quad x_{j\ell'} + w_{j\ell'} \leq x_{i\ell}$$

for all $i, j \in S : \sigma(i) < \sigma(j)$, $\ell \in \{1, \ldots, m_i\}$ and $\ell' \in \{1, \ldots, m_j\}$.

The *cost* of a feasible route is the sum of the travelling costs of the edges required to visit the customers in the specified sequence, starting and ending at the depot. Then, the 2L-CVRP consists of finding K feasible routes with minimum total cost, and such that each customer is exactly in one of them.

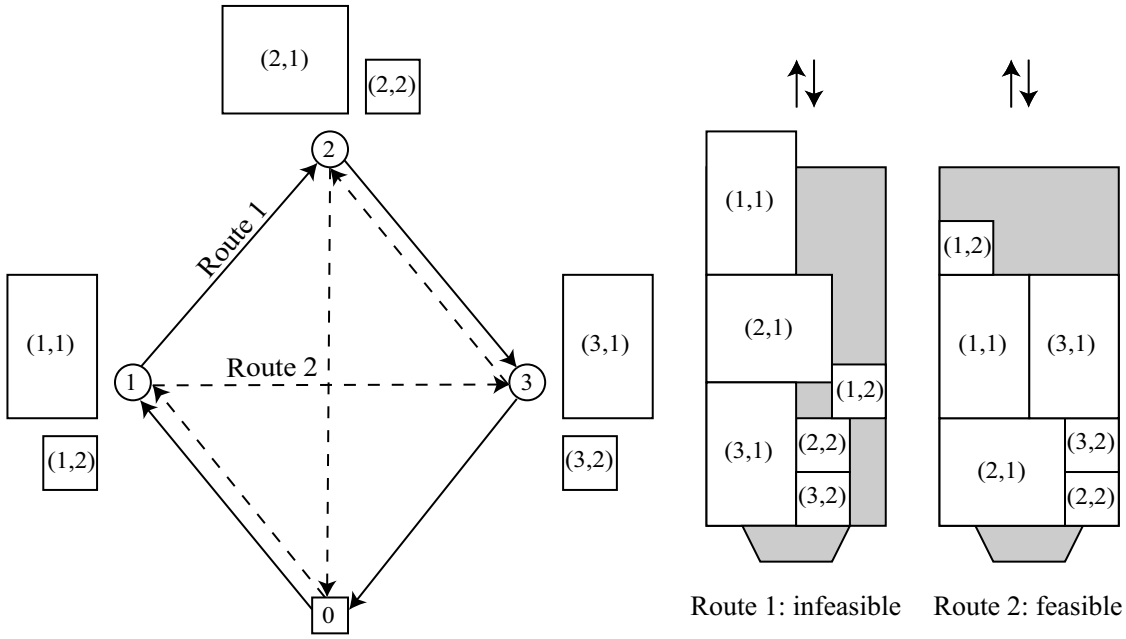

Figure 6.1: Feasible and infeasible routes for the 2L-CVRP.

An important observation when comparing the 2L-CVRP and the classical CVRP, is that the sequence σ is relevant to the feasibility of a route visiting customers in S. Figure 6.1 illustrates two different sequences σ^1 and σ^2 associated with the same customer subset $S = \{1, 2, 3\}$. Each customer has just two items. The sizes of the items and of the loading surface are proportional to those depicted in the figure. Route (S, σ^1) is defined by $\sigma^1(1) = 1$, $\sigma^1(2) = 2$, $\sigma^1(3) = 3$, and is infeasible, since item $(2, 1)$ cannot be placed side by side with items $(1, 1)$ or $(3, 1)$. Instead, route (S, σ^2), defined by $\sigma^2(1) = 1$, $\sigma^2(2) = 3$, $\sigma^2(3) = 2$, leads to a feasible loading.

Figure 6.1 also shows that route (S, σ^{2R}) in the opposite direction of σ^2, defined by $\sigma^{2R}(1) = 2$, $\sigma^{2R}(2) = 3$, $\sigma^{2R}(3) = 1$, is also a feasible route for the 2L-CVRP. This is a general result, because whenever (S, σ) is a feasible route, then also (S, σ^R), where $\sigma^R(i) = |S| - \sigma(i) + 1$, is a feasible route. Indeed, if all items in S can be loaded and unloaded feasibly from the vehicle following the sequence σ, it would be sufficient to "reverse" the vehicle in order to obtain a feasible loading for the reversed sequence σ^R. This can be done by applying a symmetry operation with respect to line $y = H/2$, and consequently changing the coordinates (x, y) to (x^R, y^R), such that $x^R_{i\ell} = x_{i\ell}$ and $y^R_{i\ell} = H - y_{i\ell} - h_{i\ell}$. Note that, as a consequence, the opposite route of an infeasible route is also infeasible.

In the following sections we describe the overall approach used to solve the 2L-CVRP.

6.3 A Branch-and-Cut Approach

This section proposes an integer linear programming model for the 2L-CVRP using large families of linear inequalities. The model is later used to solve the problem through a branch-and-cut algorithm. This section also presents the separation procedures used to find violated inequalities of the families describing the model, and a primal heuristic to speed up the solution of the algorithm.

Given a customer subset S, we denote by $\delta(S)$ the set of edges with one endpoint in S and the other in $V \setminus S$. As usual, $\delta(i)$ is used instead of $\delta(\{i\})$. Moreover, given a feasible route (S, σ) we denote by $E(S, \sigma)$ the set of edges in such a route. More precisely $E(S, \sigma) = \{(0, \sigma(1)), (\sigma(1), \sigma(2)), \ldots, (\sigma(|S|), 0)\}$. Finally, given a customer subset S we denote by $\Sigma(S)$ the collection of sequences σ such that (S, σ) is a feasible route.

6.3.1 Problem Formulation

To model the 2L-CVRP we adapted the classical two-index vehicle flow model for the CVRP (see, e.g., Toth and Vigo (2002a)). It uses a binary variable z_e for each $e \in E$ which takes value 1 if and only if a vehicle travels along edge e in a feasible route. The resulting formulation is:

$$\min \sum_{e \in E} c_e z_e \tag{6.1}$$

$$\sum_{e \in \delta(i)} z_e = 2 \qquad \text{for all } i \in V \setminus \{0\} \tag{6.2}$$

$$\sum_{e \in \delta(0)} z_e = 2K \tag{6.3}$$

$$\sum_{e \in \delta(S)} z_e \geq 2r(S) \qquad \text{for all } S \subseteq V \setminus \{0\}, S \neq \emptyset \tag{6.4}$$

$$\sum_{e \in E(S,\sigma)} z_e \leq |S| - 1 \qquad \text{for all } (S, \sigma) \text{ such that } \sigma \notin \Sigma(S) \tag{6.5}$$

$$z_e \in \{0, 1\} \qquad \text{for all } e \in E \setminus \delta(0) \tag{6.6}$$

$$z_e \in \{0, 1, 2\} \qquad \text{for all } e \in \delta(0) \tag{6.7}$$

Constraints (6.2) and (6.3) impose the degree requirements at the customer vertices, and at the depot, respectively. Constraints (6.4), known as the *capacity-cut constraints*, impose both the solution connectivity and the feasibility with respect to Conditions 1–3 of Section 6.2. In these constraints, $r(S)$ is the minimum number of vehicles that are necessary to serve the customers in set S, and not considering the actual sequence in which they are visited (i.e., by disregarding Condition 4 introduced in Section 6.2). Constraints (6.5) consider the loading sequence requirements expressed by Condition 4 by eliminating non-feasible sequences σ associated with a customer set S. Finally, constraints (6.6) and (6.7) impose that the variables associated with the edges connecting two customers are binary, and those connecting the customers to the depot may also take value two (thus representing a single-customer route).

It is well-known that for the classical CVRP the computation of $r(S)$ is difficult, as it amounts to solving a one-dimensional Bin Packing Problem (1BPP) associated with customer set S and with respect to item weights and vehicle capacity. In our case, this is even more complex since we must also take into account the 2BPP associated with the loading of the items into the vehicles, with the additional side constraint that all the items of any given customer must be loaded into the same bin. Therefore, as it is usually done in the literature, we replace $r(S)$ by a lower bound on its value given by a simple relaxation. In particular, we used

$$r'(S) = \max\left\{\left\lceil \sum_{i\in S} d_i/D \right\rceil, \left\lceil \sum_{i\in S} a_i/A \right\rceil\right\} \tag{6.8}$$

where the first term is the well-known 1BPP continuous lower bound over the customer weights, and the second term is the 2BPP continuous lower bound over the items areas of the customers in S. Bound $r'(S)$ may be further strengthened by considering better 2BPP lower bounds, as those presented in Martello and Vigo (1998).

Formulation (6.1)–(6.7) has the disadvantage of including the large families of constraints (6.4) and (6.5). However, by applying a branch-and-cut approach, we actually do not have to include all of them explicitly in the model, but just to define suitable *separation procedures*. Given a solution z^* of a relaxed model of (6.1)–(6.7), the *separation problem* consists of either proving that all constraints in a given family are satisfied by z^*, or finding a violated one. In the next section we introduce simple separation procedures for constraints (6.4) and (6.5).

6.3.2 Separation Procedures

Due to the definition of $r(S)$, constraints (6.4) induce a separation problem which is $\mathcal{NP}$-Hard (see Naddef and Rinaldi (2002)). Therefore, it is unlikely that we can design a polynomial-time exact algorithm for determining a constraint (6.4) violated by z^* when it exists. Nevertheless, as it is done for the CVRP, it is possible to propose good heuristic procedures to possibly find such a violated constraint. To this end, we adapted some simple heuristic procedures used for the same constraints in the CVRP (see, e.g., Naddef (2002)), so as to take into account also the areas of the items. In particular, we used:

Procedure 1: Let us consider the support graph $G^* = (V, E^*)$ of z^* defined by $E^* := \{e \in E : z^*_e > 0\}$ and by a capacity z^*_e associated with edge e. For each customer i, find the min-cut $\delta(S^*)$ separating 0 from i, and such that $i \in S^*$. Then check the constraint (6.4) defined by $S := S^*$ for potential violation with respect to the relaxed right-hand side $r'(S)$.

Procedure 2: Let us consider a dummy vertex $n+1$ and define a graph $G' = (V \cup \{n+1\}, E')$ where E' contains all the edges in E^* with the same capacities as in Procedure 1, plus a new edge connecting i and $n+1$ with capacity $2d_i/D$. Find the min-cut $\delta(S')$ separating 0 from $n+1$ and such that $n+1 \in S'$, and check constraint (6.4) defined for $S := S' \setminus \{n+1\}$.

Procedure 3: The same as Procedure 2 but with capacities $2a_i/A$ on the dummy edges.

Let us now address the separation problem associated with capacity-cut constraints (6.5). In our approach we separate these constraints only when Procedures 1-3 do not return a violated constraint (6.4), and when the current solution z^* is integer. It should be observed that in this case z^* defines a solution of 2L-CVRP made up by exactly K routes so that the subsets of customers associated with each different route, as well as the visit order of the customers along the routes, are uniquely determined.

More precisely, we have the following:

Procedure 4: For each route (S, σ) in the current solution, check its feasibility with respect to the loading Conditions 2–4 of Section 6.2, by using the exact algorithm *Check-2L* to be described in Section 6.4. If route (S, σ) is not feasible, then the associated constraint (6.5) is added to the

current model. Otherwise, if all routes are feasible, the overall incumbent solution is possibly updated.

We note that, since the feasibility check of a route is actually an $\mathcal{NP}$-Complete problem, it can happen that procedure *Check-2L* must be stopped whenever a prescribed time limit is exceeded without proving either the feasibility or the infeasibility of the given route.

6.3.3 Strengthening the LP-relaxation

The extensive literature on the CVRP shows different inequalities that can be added to the linear relaxation of model (6.1)–(6.4), see, e.g., Naddef and Rinaldi (2002). An example of these inequalities is represented by the following *multistar inequalities*:

$$\sum_{e\in\delta(S)} z_e \geq \frac{2}{D}\sum_{i\in S}\left(d_i + \sum_{j\in V\setminus(S\cup\{0\})} d_j z_{\{ij\}}\right) \tag{6.9}$$

for all $S \subseteq V\setminus\{0\}$. These inequalities were introduced by Araque et al. (1990) for the CVRP with unit demands. Later, Fisher (1995), Gouveia (1995) and Achuthan et al. (1998), generalized them to the CVRP with general demands, and Blasum and Hochstättler (2000) and Letchford et al. (2002) provided polynomial separation procedures based on solving max-flow problems.

6.3.4 Branching scheme

Whenever the LP-relaxation, strengthened with the additional cuts, does not produce a 2L-CVRP solution, then a branching phase is applied. In our implementation, we used the branch-and-bound framework provided by CPLEX 8.1 solver. We conducted an extensive testing on some instances from our benchmark set in order to define the most effective branching scheme and search strategy. We also compared the different search strategies provided by CPLEX and the most effective one was a combination of the "best promising node" and depth-first search (which can be obtained by setting to 0.5 the parameter CPX_PARAM_BTTOOL).

At each node we use the separation procedures described in Section 6.3.2. In addition, the heuristic algorithm to be described in the next section is applied to the final LP relaxation, both to possibly update the incumbent solution and to detect further violated inequalities of type (6.5) to be added to the current model.

As previously mentioned, separation procedure 4 requires checking the feasibility of a given route through algorithm *Check-2L*. Since this feasibility check may be very time consuming, we introduced stopping conditions for *Check-2L* based on maximum number of backtrackings and time limit. Whenever one of these conditions is achieved we are clearly neither able to find a feasible loading of the vehicle, nor to prove the infeasibility of the route. In this case we heuristically consider the route infeasible and add the corresponding (possibly not valid) cut to the current model. We should note that this implies that the final 2L-CVRP solution is not guaranteed to be optimal. However, our extensive computational experience has shown that this is a relatively rare event.

6.3.5 Heuristic Algorithms

The performance of the overall approach is enhanced by using specific heuristic algorithms for the 2L-CVRP to both derive an initial feasible solution at the root node and to possibly update the incumbent solution at other decision nodes.

A heuristic for the 2L-CVRP should combine the routing component with the two-dimensional loading one. To this end, a set of routes is determined by using a *parametric sequential insertion heuristic* that takes into account both classical routing cost minimization, as well as two-dimensional packing insertion criteria. The obtained routes are improved by means of a two-opt exchange procedure and then checked for feasibility by using procedure *Check-2L* to be defined in Section 6.4.

The insertion algorithm operates as follows. It constructs one (possibly feasible) route at a time. The route is initialized by connecting the depot to the most distant unrouted customer. The next vertex to be inserted is selected as the one minimizing a modified insertion cost $\widetilde{\Gamma}_i$, that takes into account the simple routing insertion cost, the residual area in the vehicle and the residual weight capacity. More precisely, given a customer i and the current route (S, σ), let us define

$A^{res} = (A - \sum_{i \in S} a_i)$, the residual loading area in the vehicle;

$D^{res} = (D - \sum_{i \in S} d_i)$, the residual weight capacity in the vehicle;

Γ_i , the cost of the best insertion of customer i in the route (S, σ).

The modified insertion cost of customer i in route (S, σ) is then:

$$\widetilde{\Gamma}_i = \alpha\Gamma_i + \beta(A^{res} - a_i) + \gamma(D^{res} - d_i) \tag{6.10}$$

where α, β, γ are suitably defined non-negative parameters such that $\alpha+\beta+\gamma = 1$. In addition, $\widetilde{\Gamma}_i = \infty$ whenever either $a_i > A^{res}$ or $d_i > D^{res}$. It should be observed that parameters α, β and γ allow the combination of classical routing and packing insertion criteria. For example, when $(\beta = \gamma = 0)$ we have the *cheapest insertion* rule used for the CVRP (see, e.g., Mole and Jameson (1976)). On the other hand, when $\alpha = \gamma = 0$ (or alternatively $\alpha = \beta = 0$) the insertion criterion turns out to be the well-known *first fit* rule used in the bin packing context (see, e.g., Johnson (1973)).

As soon as the current vehicle may not accommodate further customers and the total number of used routes is smaller than K, a new route is initialized and the insertion process is iterated. If all customers were inserted in the K routes, then a first-improvement descent procedure based on a two-opt neighborhood is applied to possibly improve the overall solution cost.

This constructive procedure is repeated ten times with different values of parameters α, β and γ, and the best solution is considered. If the resulting solution value is smaller than the actual best known feasible 2L-CVRP solution, then the actual feasibility with respect to the loading of each route is checked through procedure *Check-2L*, to be described in Section 6.4. The procedure is called with a limit on the CPU time and on the maximum number of backtrackings. If all routes are feasible, then the current best known 2L-CVRP solution is updated. Otherwise, for each route that is proved to be infeasible the corresponding constraint is generated and added to the model.

The above heuristic procedure is executed at the beginning of the branch-and-cut algorithm to possibly produce a first feasible 2L-CVRP solution. In order to exploit the information of the current LP-relaxation at each branch-decision node, this heuristic scheme is also executed with modified edge costs defined as

$$c'_e = c_e(1 - z^*_e/2) \tag{6.11}$$

where, for each edge $e \in E$, c_e is the original cost and z^*_e is the value of the associated variable in the current LP solution.

6.4 A Branch-and-Bound Algorithm for the Loading-Check Problem

The overall solution approach for the 2L-CVRP outlined in the previous section strongly relies on a procedure to check the existence of a feasible loading pattern for a given route (S, σ), according to Conditions 1–4 of Section 6.2. This problem hereafter is referred to as 2L. It is easy to see that 2L is $\mathcal{NP}$-Complete in the strong sense, since it generalizes other two-dimensional single-bin filling problems as the one described in Martello et al. (2000).

In this section we describe an algorithm, called *Check-2L*, to exactly solve problem 2L. The algorithm is based on a branch-and-bound approach, and extends the one proposed by Martello et al.

(2000) for a single-bin filling problem arising within the solution of the three-dimensional Bin Packing Problem (3BPP).

At the root node of the branch-and-bound search tree we compute lower bounds to possibly avoid entering the actual enumeration scheme. To this end, we relax the sequential loading requirements and use the lower bounds proposed by Martello and Vigo (1998) for the 2BPP. If the largest lower bound value is greater than one, then clearly the 2L instance is infeasible. Otherwise, the items are sorted in reversed customer loading sequence (i.e., the first customer to be visited is the last one to be loaded on the vehicle), and by non-increasing width within each customer (breaking ties by non-increasing height).

We then use a simple heuristic procedure, derived from the classical bottom-left approach (see, e.g., Berkey and Wang (1987)), to possibly detect a feasible single-bin loading pattern. The heuristic places one item at a time (in the given order) in the lowest feasible position in the bin and left-justified, checking that the Conditions 2–4 are not violated.

Whenever the lower and upper bounds did not prove the feasibility or infeasibility of the route (S, σ), an implicit enumeration scheme is used. The scheme starts with an empty solution and generates one descendant node for each item, in the given order, by placing it in the bottom-left position (i.e., at coordinates $(0, 0)$) of the loading surface.

At each subsequent level of the search tree, a new item is placed in a feasible position of the loading surface. Given a partial solution associated with a node of the search tree, where some items have been placed, descendant nodes are generated, one for each non-placed item and for each feasible placement position.

Scheithauer (1997) has shown that the set of feasible placement positions for the two-dimensional packing can be limited to the finite set of the *contour points* of the *contour* associated with the already placed items (similar results were described by Martello et al. (2000) for the 3BPP). In the 2L case, we may further reduce the set of feasible placement positions by taking into account the customer sequencing constraints. Indeed, an item may only be placed at a given contour point in the loading surface, if it does not prevent the unloading of other items belonging to customers visited earlier in the route. We illustrates this reduction by means of the simple example depicted in Figure 6.2, in which we consider a vehicle surface with $W = 20$ and $H = 30$.

The figure gives a partial loading associated with a route (S, σ), with $S = \{1, 2, 3\}$ and $\sigma(1) = 1$, $\sigma(2) = 2$ and $\sigma(3) = 3$, where four items of these customers have already been placed. Given this partial loading, the current contour line is indicated in the figure by a thick dashed line, and the set of contour points is made up by the two positions $(0, 20)$ and $(7, 16)$, indicated by a circle. We note that, as explained in Scheithauer (1997), we can consider the area above item $(2, 1)$ and below the contour line as "trapped" and not available for placement of items in the descendant nodes, thus allowing for a reduction of the enumeration tree size. In addition, we note that, because of the sequential loading requirements, at contour point $(0, 20)$ we may place either items of customer 1 with no restriction, or items of customer 2 with $w_{2\ell} \leq 14$, or items of customer 3 with $w_{3\ell} \leq 7$. Similarly, at contour point $(7, 16)$ we may place either items of customer 1, or items of customer 2 with $w_{2\ell} \leq 7$, and no item of customer 3.

Whenever an item is placed, we may perform feasibility tests and compute lower bounds to possibly fathom the current node of the enumeration tree. Martello et al. (2000) used for the single-bin 2BPP and 3BPP, a simple bound that estimates the total area, T, that can be placed given a partial allocation, as the sum of the areas of the already placed items, plus the total available area above the contour line. If T is not greater than the largest total placed area T^* found so far, then the current node can be clearly fathomed since it may not lead to an improved solution.

In our case, we may define additional simple feasibility tests by considering that the area above the contour line is not necessarily available for all the unplaced items. Indeed, we may divide this area in *regions* and for each of these regions we can define the customers that may use such region to place their items. Figure 6.2 illustrates this by showing the three regions in which is divided the area above the contour line associated with the partial loading. Then we may fathom the current node as soon as the available area for a given customer is smaller than that of his unplaced items. In addition, we may fathom the node if, for a given customer, the total width of adjacent regions that may accommodate

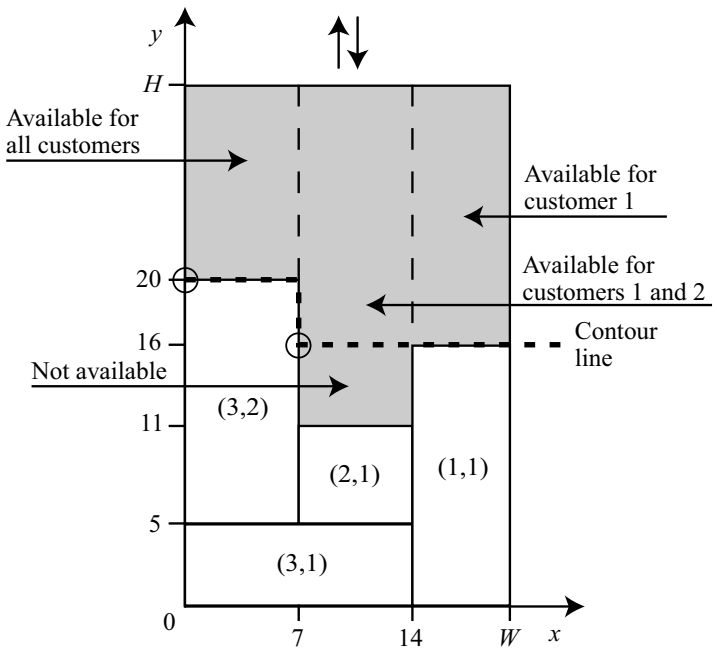

Figure 6.2: A partial loading with areas available for the next items to be placed. Route (S, σ) has $S = \{1, 2, 3\}$ and $\sigma(1) = 1$, $\sigma(2) = 2$ and $\sigma(3) = 3$.

his items is smaller than the width of his widest unplaced item, or when the maximum height of the available region is smaller than the height of his tallest unplaced item.

As previously mentioned, procedure *Check-2L* may consume a large portion of the total time required to solve the overall 2L-CVRP. Moreover, since the same route (S, σ) can appear in several different solutions generated by the branch-and-cut algorithm for the 2L-CVRP, we store the routes already tested by procedure *Check-2L* in a pool structure, thus avoiding to solve the same 2L instance twice.

6.5 Computational Results

The algorithm has been coded in C and run on a Pentium IV 1700 MHz. The branch-and-cut was embedded in the framework provided by the CPLEX 8.1 solver.

In order to test the performance of the algorithm, we defined a set of test instances by extending to the 2L-CVRP some CVRP instances from the literature (see Reinelt (1991) or Toth and Vigo (2002a) for a definition of these CVRP instances). Each 2L-CVRP instance takes from the corresponding CVRP one the complete graph, the weights d_i associated with the customers and the weight capacity D of the vehicles. The costs of the arcs are integer values obtained by truncating the value of the distances between the corresponding endpoints.

We considered five classes of instances (see Table 6.1), each corresponding to a different way used to generate the items associated with each customer. The first class corresponds to the original CVRP instance (obtained by assigning to each customer a single item having both sizes equal to 1, and by setting $W = H = n$). As to the remaining classes, the item and bin sizes are generated in a similar way as in the 2BPP literature. For each class $r = 2, \ldots, 5$, the size of the loading surface of the vehicles is set to $W = 20$ and $H = 40$, and the number of items generated for each customer is a uniformly random integer value in the interval $[1, r]$. For each of these items, a possible shape, namely *Vertical*,

Homogeneous or *Horizontal*, is selected with equal probability, and the corresponding item sizes are randomly generated in the intervals given in Table 6.1.

Table 6.1: Classes used for the generation of the items.

		Vertical		*Homogeneous*		*Horizontal*	
Class	m_i	$h_{i\ell}$	$w_{i\ell}$	$h_{i\ell}$	$w_{i\ell}$	$h_{i\ell}$	$w_{i\ell}$
1	1	1	1	1	1	1	1
2	[1, 2]	$[\frac{4H}{10}, \frac{9H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{2H}{10}, \frac{5H}{10}]$	$[\frac{2W}{10}, \frac{5W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{4W}{10}, \frac{9W}{10}]$
3	[1, 3]	$[\frac{3H}{10}, \frac{8H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{2H}{10}, \frac{4H}{10}]$	$[\frac{2W}{10}, \frac{4W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{3W}{10}, \frac{8W}{10}]$
4	[1, 4]	$[\frac{2H}{10}, \frac{7H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{H}{10}, \frac{4H}{10}]$	$[\frac{W}{10}, \frac{4W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{2W}{10}, \frac{7W}{10}]$
5	[1, 5]	$[\frac{H}{10}, \frac{6H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{H}{10}, \frac{3H}{10}]$	$[\frac{W}{10}, \frac{3W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{W}{10}, \frac{6W}{10}]$

It should be observed that in these instances there is no correlation between the weight of the items and their overall area. In preliminary experiments, we also extensively generated correlated instances, and they resulted easier to solver than the uncorrelated ones (see Iori (2004) for more details).

For each instance, the number K of available vehicles is determined as follows. For instances of Class 1, K is set equal to the corresponding value in the original CVRP problem. For the other classes, we heuristically solve a 2BPP associated with the items, by using an adaptation of the heuristic described in Martello et al. (2000) for the 3BPP. Then K is set to the maximum value between the number of bins in the heuristic solution and the number of available vehicles in the original CVRP instance. It should be noted that, since in this heuristic approach the clustering and sequential loading constraints are not explicitly taken into account, the resulting K may be smaller that the actual minimum number of vehicles needed to load all the items, thus leading to an infeasible 2L-CVRP instance. To this end, whenever the algorithm terminates proving that no feasible solution exists for a given K, we increase its value by one unit and try to solve the new instance. (This situation never occurred in our experiments: the exact algorithm was always able to find a feasible solution for all generated instances.) We finally note that, as it is frequently done in the routing context, we did not allow single-customer routes.

We considered 12 CVRP instances and, for each of them, we generated one instance for each class, obtaining in total 60 instances, with up to 35 customers and 114 items. The original CVRP instances and the complete set of 2L-CVRP instances can be downloaded from:

`http://www.or.deis.unibo.it/research.html`.

We imposed a CPU time limit of 86400 seconds (i.e., one day) to the overall algorithm. The branch-and-bound algorithm of Section 6.4 was allowed a maximum time of 7200 seconds (i.e., 2 hours) for each execution of *Check-2L* requested by the Separation Procedure 4 of Section 6.3.2, and a maximum time of 100 seconds as well as a maximum number of 10000 backtrackings for each execution requested by the heuristic algorithm of Section 6.3.5.

The results of our computational experience are summarized in Tables 6.2, 6.3 and 6.4. For each instance, *Name* and *Class* give the name of the original CVRP instance and the class used for the creation of the items, respectively. Then, n represents the number of customers, M the total number of items and K the number of available vehicles.

The performance of the algorithm in the solution of each instance is summarized in three groups of columns denoted as *Loading*, *Routing*, and *Overall*. For what concerns the loading group, *Pool* reports the size of the pool of the feasible and infeasible loading constraints kept in memory, and N_{BB} gives the total number of times algorithm *Check-2L* was called by the Separation Procedure 4 of Section

6.3.2. $Fails$ gives the number of calls in which *Check-2L* was not able either to find a feasible loading for the subproblem, or to prove its infeasibility. Finally T_{load} represents the total CPU time in seconds used by the loading-related procedures. It is worth noting that the pool is filled with N_{BB} (feasible or infeasible) entries by the Separation Procedure 4, and $Pool - N_{BB}$ entries by the heuristic approach of Section 6.3.5.

In the routing group, $Cuts$ gives the number of cuts obtained through Separation Procedures 1–4 and added to the reduced model (not including default cuts added by CPLEX), $\%_{gap}$ reports the percentage gap between the value z of the best feasible solution found and the final lower bound LB of the root node (computed as $\%_{gap} = 100(z - LB)/LB$), and T_{rout} gives the total time spent by the routing-related procedures. Finally, in the overall group, z gives the value of the best feasible solution found, T_z the time required to obtain such a solution (i.e., the time in which the last update of the incumbent solution was made), and T_{tot} reports the computing time used by the overall algorithm (i.e., $T_{tot} = T_{load} + T_{rout}$). All the times are expressed in seconds on a PC Pentium IV 1700 Mhz. We note that a solution is proved to be optimal if and only if it achieves $Fails = 0$ and $\%_{gap} = 0$. These optimal values are depicted in bold characters.

By observing Tables 6.2, 6.3 and 6.4, we may see that the proposed algorithm was able to solve all instances with up to 25 customers within moderate computing time (always smaller than one hour), whereas a few larger instances were not solved within the overall time limit (one day). The size of the solved instances is consistent with the results reported in the literature for the two separate problems that are combined into the 2L-CVRP. Indeed, a basic branch-and-cut approach for the CVRP using just simple separation procedures for constraints (6.4) may hardly solve instances with more than 30 customers, whereas state-of-the-art branch-and-bound algorithms for 2BPP (see, e.g., Martello and Vigo (1998)) may solve loading instances similar to those generated in our work with at most 80–100 items.

The effect of the loading component of the 2L-CVRP is witnessed by the fact that both K and the optimal solution value are in general larger than those of the original CVRP instance. The few exceptions to this increase in the routing cost are represented by instances of Class 5, in which the item sizes are relatively small.

Procedure *Check-2L* proved to be particularly efficient for the resolution of the 2L. For the complete set of instances, *Check-2L* failed only a few times to prove the feasibility or infeasibility of the loading. These exceptions arise especially for instances of Class 5, and this is explained by the fact that this class is characterized by a large number of items to be loaded into each vehicle. This is a typical feature of very difficult 2BPPs. As a consequence, for each CVRP instance, T_{load} augments from Class 2 to Class 5, and in some cases can absorb a very large part of the computing time. We also note that the final solution of instance $E030 - 03g$ Class 5 was not proved to be optimal, due to the failures of *Check-2L* that may result in the addition of possibly not valid cuts (see Section 6.3.4).

When addressing larger CVRP instances, T_{rout} increases consistently. This is however not surprising, since it is well known that the separation of constraints (6.4) does not suffice for the quick solution of CVRPs of these sizes through branch-and-cut. All the instances of Class 1 are solved to optimality. Among the other classes (where the routing does not absorb the entire computing time granted to the algorithm), in five cases the branch-and-cut approach does not manage to explore the whole tree, and gives back a value of $\%_{gap}$ that can be very high. The number of cuts added during the execution of the algorithm may be quite large and increases with the size of the instance.

6.6 Conclusions

In this paper we examined the problem of finding the optimal routes for K vehicles used to serve the demand of n customers, consisting of a set of rectangular items, each characterized by its weight, height and width. Each vehicle has a weight capacity limit and a rectangular loading surface, characterized by a height and a width. This problem, denoted as 2L-CVRP, combines the classical well-known and notoriously difficult Capacitated Vehicle Routing Problem and two-dimensional Bin Packing Problem, and leads to a very complicated overall combinatorial problem.

We presented an integer linear programming model for the 2L-CVRP. In particular, the model contains two families of constraints, the first used to impose the weight capacity-cut constraints and the second used to forbid infeasible loading patterns. Since both families involve an exponentially-growing number of constraints, we adopted a branch-and-cut approach for the exact solution of the model. We presented heuristic separation procedures used to possibly detect violated constraints of both families. In particular, for the loading-related constraints we developed a specialized branch-and-bound approach used to check the feasibility of the loading of a given subset of items into a single vehicle.

The proposed approach was successfully tested on instances derived from classical CVRP ones, involving up to 35 customers and more than 100 items. This set of instances is particularly challenging, since both the maximum number of customers and items are close to the maximum size of the problems that may be solved by similar codes for the two combinatorial problems combined in the 2L-CVRP. Indeed, simple branch-and-cut approaches using just a fractional capacity-cut constraints separation may hardly solve CVRP instances with more that 25-30 customers, whereas state-of-the-art 2BPP codes may solve instances with no more than 80-100 items. Within 24 hours of computing time, the proposed algorithm was able to solve all the instances of the test bed except five cases. Moreover, all the instances with at most 25 customers were solved in less than one hour of computing time.

Table 6.2: Overall performance of the algorithm.

					Loading				*Routing*			*Overall*		
Name	*Class*	n	M	K	*Pool*	N_{BB}	*Fails*	T_{load}	*Cuts*	$\%_{gap}$	T_{rout}	z	T_{heur}	T_{tot}
$E016-03m$	1	15	15	3	18	**100**	32	0.02	252	**0.0**	3.84	**273**	3.80	3.86
	2	15	24	3	53	**100**	120	8.64	828	**0.0**	60.18	**285**	51.38	68.81
	3	15	31	3	17	**100**	71	11.93	474	**0.0**	9.49	**280**	17.06	21.42
	4	15	37	4	9	**100**	18	2.28	29	**0.0**	0.08	**288**	1.80	2.36
	5	15	45	4	5	**100**	5	53.03	4	**0.0**	0.05	**279**	53.08	53.08
$E016-05m$	1	15	15	5	8	**100**	26	0.02	116	**0.0**	0.53	**329**	0.19	0.55
	2	15	25	5	26	**100**	46	0.08	387	**0.0**	11.88	**342**	11.08	11.95
	3	15	31	5	3	**100**	33	0.61	380	**0.0**	7.53	**347**	6.39	8.14
	4	15	40	5	5	**100**	32	20.80	219	**0.0**	2.62	**336**	22.42	23.42
	5	15	48	5	8	**100**	19	28.80	116	**0.0**	0.56	**329**	29.22	29.36
$E021-04m$	1	20	20	4	38	**100**	64	0.03	362	**0.0**	15.58	**351**	14.92	15.61
	2	20	29	5	185	**100**	561	3.74	970	**0.0**	62.04	**389**	32.59	65.78
	3	20	46	5	30	**100**	39	4.04	142	**0.0**	2.02	**387**	6.03	6.06
	4	20	44	5	36	**100**	79	38.15	130	**0.0**	1.21	**374**	39.28	39.36
	5	20	49	5	10	**100**	18	0.11	46	**0.0**	0.11	**369**	0.17	0.22
$E021-06m$	1	20	20	6	8	**100**	37	0.00	111	**0.0**	0.41	**423**	0.38	0.41
	2	20	32	6	7	**100**	32	0.03	248	**0.0**	5.44	**434**	5.30	5.47
	3	20	43	6	13	**100**	70	3.06	311	**0.0**	5.09	**432**	5.23	8.16
	4	20	50	6	24	**100**	93	2.58	520	**0.0**	23.80	**438**	23.95	26.38
	5	20	62	6	10	**100**	48	44.28	122	**0.0**	0.42	**423**	23.00	44.70
$E022-04g$	1	21	21	4	8	**100**	16	0.00	41	**0.0**	0.08	**367**	0.08	0.08
	2	21	31	4	42	**100**	84	1.78	249	**0.0**	6.58	**380**	7.98	8.36
	3	21	37	4	10	**100**	30	1.77	52	**0.0**	0.14	**373**	1.81	1.91
	4	21	41	4	29	**100**	120	45.52	351	**0.0**	4.78	**377**	50.20	50.30
	5	21	57	5	26	**100**	51	2928.05	44	**0.0**	0.22	**389**	2928.20	2928.27
$E022-06m$	1	21	21	6	26	**100**	47	0.02	464	**0.0**	10.67	**488**	5.89	10.69
	2	21	33	6	46	**100**	95	0.83	1227	**0.0**	145.03	**491**	145.83	145.86
	3	21	40	6	34	**100**	66	2.34	1154	**0.0**	148.03	**496**	135.41	150.37
	4	21	57	6	8	**100**	55	5.25	598	**0.0**	11.31	**489**	14.12	16.56
	5	21	56	6	32	**100**	68	0.27	515	**0.0**	13.08	**488**	10.76	13.34

Table 6.3: Overall performance of the algorithm.

					Loading				*Routing*				*Overall*	
Name	*Class*	n	M	K	*Pool*	N_{BB}	*Fails*	T_{load}	*Cuts*	$\%_{gap}$	T_{rout}	z	T_{heur}	T_{tot}
$E023-03g$	1	22	22	3	3	**100**	3	0.00	30	**0.0**	0.02	**558**	0.02	0.02
	2	22	32	5	109	**100**	233	15.86	445	**0.0**	16.61	**724**	32.05	32.47
	3	22	41	5	20	**100**	48	4.11	47	**0.0**	0.28	**698**	3.19	4.39
	4	22	51	5	44	**100**	70	2595.55	184	**0.0**	1.72	**714**	2596.83	2597.27
	5	22	55	6	38	**100**	52	747.06	20	**0.0**	0.11	**742**	738.94	747.17
$E023-05s$	1	22	22	5	10	**100**	23	0.00	6	**0.0**	0.05	**657**	0.03	0.05
	2	22	29	5	466	**100**	630	3.29	889	**0.0**	88.53	**720**	75.94	91.81
	3	22	42	5	54	**100**	228	67.15	511	**0.0**	5.80	**730**	70.03	72.95
	4	22	48	5	10	**100**	22	14.05	37	**0.0**	0.08	**701**	7.44	14.12
	5	22	52	6	6	**100**	6	1128.86	8	**0.0**	0.00	**721**	1128.86	1128.86
$E026-08m$	1	25	25	8	9	**100**	50	0.00	561	**0.0**	31.91	**609**	6.17	31.91
	2	25	40	8	36	**100**	144	0.11	1610	**0.0**	460.41	**612**	453.52	460.52
	3	25	61	8	37	**100**	84	1.96	1175	**0.0**	192.36	**615**	164.83	194.31
	4	25	63	8	41	**100**	103	9.91	1613	**0.0**	1583.44	**626**	852.14	1593.34
	5	25	91	8	20	**100**	78	33.88	639	**0.0**	35.30	**609**	47.39	69.17
$E030-03g$	1	29	29	3	29	**100**	49	0.02	5130	1.12	86400.50	524	1226.17	86400.52
	2	29	43	6	2876	**100**	4412	241.97	11668	13.21	86159.18	774	48373.34	86401.15
	3	29	49	6	363	**100**	785	269.16	1545	**0.0**	2880.84	**638**	2304.46	3149.99
	4	29	72	7	224	**100**	596	11243.17	1715	**0.0**	1453.12	**738**	12671.31	12696.29
	5	29	86	7	69	88	343	70158.76	417	**0.0**	149.26	706	48220.23	70308.02
$E030-04s$	1	29	29	4	8	**100**	12	0.02	27	**0.0**	0.13	**500**	0.13	0.14
	2	29	43	6	2210	**100**	3573	109.72	10089	19.04	86290.74	789	99.80	86400.46
	3	29	62	7	760	**100**	1300	576.53	4915	4.85	85823.79	763	63747.77	86400.32
	4	29	74	7	1202	**100**	2836	20086.61	13879	14.54	66314.22	881	85181.52	86400.83
	5	29	87	7	42	76	94	86397.05	65	2.09	3.02	695	64392.94	86400.06
$E031-09h$	1	30	30	9	15	**100**	80	0.55	4161	1.50	86399.79	599	50093.46	86400.34
	2	30	50	9	10	**100**	211	6.31	7296	6.78	86394.36	625	75.60	86400.67
	3	30	56	9	13	**100**	81	1.39	3976	0.95	86399.19	597	36171.03	86400.58
	4	30	82	9	29	**100**	335	132.18	7857	6.97	86267.99	624	86321.63	86400.16
	5	30	101	9	24	**100**	105	7570.50	4524	2.53	78829.86	602	22352.36	86400.36

Table 6.4: Overall performance of the algorithm.

					Loading				*Routing*				*Overall*	
Name	*Class*	n	M	K	*Pool*	N_{BB}	*Fails*	T_{load}	*Cuts*	$\%_{gap}$	T_{rout}	z	T_{heur}	T_{tot}
$E033-03n$	1	32	32	3	29	**100**	38	0.08	289	**0.0**	14.44	**1991**	13.18	14.52
	2	32	44	7	4491	**100**	2696	116.95	15313	25.94	172685.26	3523	10288.85	86401.33
	3	32	56	7	324	**100**	738	157.20	2238	**0.0**	31967.35	**2570**	3087.87	32124.54
	4	32	78	7	289	**100**	1587	14004.82	3391	1.30	72395.55	2673	34999.17	86400.38
	5	32	102	8	52	79	119	86386.10	218	6.47	14.00	2807	83628.40	86400.10
$E033-04g$	1	32	32	4	38	**100**	58	0.58	5200	2.58	86399.85	827	25734.79	86400.43
	2	32	47	7	2319	**100**	6460	477.60	18283	18.39	85923.24	1459	80521.15	86400.84
	3	32	57	7	216	**100**	627	81.65	5767	3.75	86318.44	1211	42204.75	86400.09
	4	32	65	7	94	**100**	377	25093.92	1076	**0.0**	448.70	**1166**	25391.47	25542.61
	5	32	87	8	21	48	87	86398.60	78	15.34	1.48	1504	80511.08	86400.08
$E033-05s$	1	32	32	5	45	**100**	61	0.00	333	**0.0**	17.84	**907**	17.81	17.84
	2	32	48	6	426	**100**		621.82	18919	13.05	85779.62	1203	386.89	86401.44
	3	32	59	6	667	**100**	4979	4338.36	19424	23.24	82062.85	1405	15880.73	86401.21
	4	32	84	8	155	97	470	84777.90	3116	2.90	1622.37	1358	55614.54	86400.27
	5	32	114	8	25	56	164	86354.70	770	4.96	45.41	1390	59867.80	86400.10
$E036-11h$	1	35	35	11	63	**100**	122	0.45	2289	**0.0**	7086.26	**682**	7086.48	7086.71
	2	35	56	11	24	**100**	123	0.06	1899	**0.0**	3374.70	**682**	1766.95	3374.76
	3	35	74	11	1	**100**	79	0.22	1581	**0.0**	3853.13	**682**	345.15	3853.35
	4	35	93	11	34	**100**	133	450.61	2804	**0.0**	32941.22	**691**	33375.93	33391.84
	5	35	114	11	6	**100**	48	0.72	1417	**0.0**	2783.28	**682**	339.03	2783.99
$E041-14h$	1	40	40	14	0	**100**	66	0.02	5612	4.86	86400.26	859	493.15	86400.27
	2	40	60	14	0	**100**	74	1.35	6767	5.97	86399.06	866	283.79	86400.41
	3	40	73	14	0	**100**	79	0.09	5045	3.45	86399.93	850	655.98	86400.02
	4	40	96	14	0	**100**	74	0.11	5471	3.76	86400.27	853	102.91	86400.38
	5	40	127	14	0	**100**	83	1.70	4408	2.42	86398.81	845	263.56	86400.52

Chapter 7

A Tabu Search approach for the 2L-CVRP

[1]

The chapter addresses the well-known *Capacitated Vehicle Routing Problem* (CVRP), in the special case where the demand of a customer consists of a certain number of two-dimensional weighted items. The problem calls for the minimization of the cost of transportation needed for the delivery of the goods demanded by the customers, and operated by a fleet of vehicles based at a central depot. In order to accommodate all items on the vehicles, a feasibility check of the two-dimensional packing (2L) must be executed on each vehicle. The overall problem, denoted as 2L-CVRP, is NP-hard and particularly difficult to solve in practice. We propose a Tabu Search heuristic, in which 2L is solved through heuristics, lower bounds and a truncated branch-and-bound procedure. The effectiveness of the algorithm is demonstrated through extensive computational experiments.

7.1 Introduction

One of the most frequently studied combinatorial optimization problems is the *Capacitated Vehicle Routing Problem* (CVRP), which calls for the optimization of the delivery of goods, demanded by a set of clients, and operated by a fleet of vehicles of limited capacity based at a central depot. The application of this model to real world problems is limited by the presence of many additional constraints. In particular, in CVRP client demands are expressed by an integer value, representing the total weight of the items to be delivered, while in real-world instances demands consist of lots of items characterized both by a weight and a three-dimensional volume.

The problem of loading or unloading these three-dimensional general-shape lots to and from vehicles is not trivial and can lead to impractical models for which it is impossible to design effective algorithms. It is, however, possible to simplify the problem. First, it is quite usual to place items into rectangular-shaped boxes. Additionally, in many cases boxes (e.g., pallets) cannot be stocked one over the other because of their fragility or weight. We thus restrict our study to the loading of two-dimensional, weighted and rectangular-shaped items.

The resulting problem, which combines the feasible loading or unloading of such items into vehicles and the minimization of transportation costs, is particularly relevant for freight distribution companies. From an algorithmic point of view, the combination of these two areas of combinatorial optimization

[1] The results of this chapter appears in: M. Gendreau, M. Iori, G. Laporte and S. Martello, A Tabu Search approach to vehicle routing problems with two-dimensional loading constraints, technical report OR/04/1, DEIS, Università di Bologna, 2004 [60].

(vehicle routing and two-dimensional packing) is challenging. We consider two variants of the problem (see Section 7.2), one of which was first addressed by Iori, Salazar González and Vigo [98]. Following the notation of these authors, the problem will be denoted here as 2L-CVRP (*Two-Dimensional Loading Capacitated Vehicle Routing Problem*).

The vehicle routing problem has been deeply investigated since the seminal work of Dantzig and Ramser [34]. Branch-and-cut algorithms are nowadays the best exact approaches (see, e.g., Toth and Vigo [144]), but are able to systematically solve to optimality only relatively small instances. The problem of loading two-dimensional items into the vehicle is closely related to various packing problems. In particular:

- the *Two-Dimensional Bin Packing Problem* (2BPP): pack a given set of rectangular items into the minimum number of identical rectangular bins. Exact approaches for 2BPP are generally based on branch-and-bound techniques and are able to solve instances with up to 100 items (but some instances with 20 items remain unsolved). Recent exact algorithms and lower bounds have been proposed by Martello and Vigo [120], Fekete and Schepers [53], Boschetti and Mingozzi [18, 19] and Pisinger and Sigurd [128].
- the *Two-Dimensional Strip Packing Problem* (2SP): pack a given set of rectangular items into a strip of given width and infinite height so as to minimize the overall height of the packing. An exact algorithm proposed by Martello, Monaci and Vigo [115] can solve instances with up to 200 items, although some instances with 30 items are still unsolved.

Both the CVRP and the 2BPP are strongly NP-hard and very difficult to solve in practice. The same holds for the 2L-CVRP. To our knowledge, the only available solution methodology for the 2L-CVRP is the exact approach developed by Iori, Salazar González and Vigo [98]: it consists of a branch-and-cut algorithm that handles the routing aspects of the problem, combined with a nested branch-and-bound procedure to check the feasibility of the loadings. The algorithm can solve instances involving up to 30 clients and 90 items. Since these limits are not reasonable for real-life problems, it is natural to consider heuristic and metaheuristic techniques. In this paper we propose a Tabu Search algorithm for the 2L-CVRP.

Several metaheuristic approaches are available for CVRP (see Glover et al. [30] for a recent survey): Tabu Search, Genetic Algorithms, Simulated and Deterministic Annealing, Ant Colonies and Neural Networks. Very good results were obtained through Tabu Search (Cordeau and Laporte [31]). Metaheuristic techniques have also been widely applied to packing problems. 2BPP was solved through Simulated Annealing by Dowsland [47] and through Tabu Search by Lodi, Martello and Vigo [112]. Tabu Search and Genetic Algorithms for 2SP were developed by Iori, Martello and Monaci [97].

The success of Tabu Search both for routing and packing problems influenced our choice in favor of such technique for 2L-CVRP. An additional consideration is that even the problem of finding a feasible solution to 2L-CVRP (i.e., a partition of the clients into subsets that satisfy the weight capacity of the vehicles and for which a feasible two-dimensional loading exists) is already strongly NP-hard and very difficult to solve in practice. Techniques based on populations of solutions, such as Genetic Algorithms and Scatter Search, would be forced to deal with a high number of infeasible solutions and hence do not seem very promising.

In Section 7.2 we introduce our notation and give a detailed description of the addressed problems. In Section 7.3 we present the proposed Tabu Search approach. The intensification phase, which constitutes a very relevant component of the algorithm, is described in detail in Section 7.4. Computational results on small-size and large-size instances are given in Section 7.5.

7.2 Problem Description

We are given a complete undirected graph $G = (V, E)$, where V is a set of $n+1$ vertices corresponding to the depot (vertex 0) and to the clients (vertices $1, \ldots, n$), and E is the set of edges (i, j), each having an associated cost c_{ij}. There are v identical vehicles, each having a weight capacity D and a rectangular loading surface of width W and height H. Let $A = W \times H$ denote the loading area. The

demand of client i ($i = 1, \ldots, n$) consists of m_i items of total weight d_i: item $I_{i\ell}$ ($\ell = 1, \ldots, m_i$) has width $w_{i\ell}$ and height $h_{i\ell}$. Let $a_i = \sum_{\ell=1}^{m_i} w_{i\ell} h_{i\ell}$ denote the total area of the lot demanded by client i.

We assume that the items have a fixed orientation, i.e., they must be packed with the w-edge (resp. the h-edge) parallel to the W-edge (resp. the H-edge) of the loading surface. This restriction is quite usual in logistics when pallet loading is considered, since there can be a fixed orientation for the entering of the forks that prohibits rotations during the loading and unloading operations. We also assume, as is usual for VRP, that each client must be served by a single vehicle. When a vehicle k is assigned a tour that includes a client set $S(k) \subseteq \{1, \ldots, n\}$, the two following constraints must be satisfied:

Weight constraint: the total weight $\sum_{i \in S(k)} d_i$ must not exceed the vehicle capacity D;

Loading constraint: there must be a feasible (non overlapping) loading of the items in

$$I(k) = \bigcup_{i \in S(k)} \bigcup_{\ell \in \{1,\ldots,m_i\}} I_{i\ell} \tag{7.1}$$

into the $W \times H$ loading area.

The objective is to find a partition of the clients into at most v subsets and, for each subset $S(k)$, a route starting and ending at the depot (vertex 0) such that both conditions above hold, and the total cost of the edges in the routes is a minimum. An example is depicted in Figure 7.1.

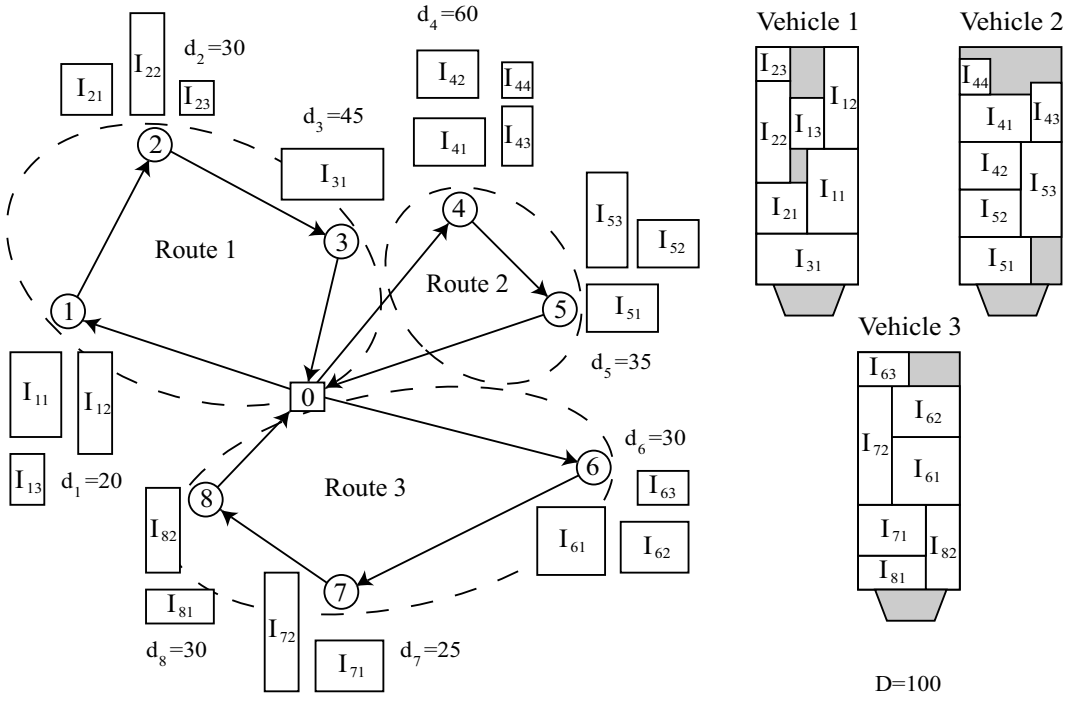

Figure 7.1: *Example of the 2L-CVRP.*

We will consider two variants of the problem: the *Unrestricted* 2L-CVRP and the *Sequential* 2L-CVRP. In the former one no additional constraint is imposed, while in the latter we also impose to each vehicle, say k, the following constraint:

Sequence constraint: the vehicle loading must be such that when client $i \in S(k)$ is visited, the items of his lot, $\bigcup_{\ell \in \{1,\ldots,m_i\}} I_{i\ell}$, can be downloaded through a sequence of straight movements (one

per item) parallel to the H-edge of the loading area.

In other words, for any item $I_{i\ell}$, no item demanded by a client visited later on in the tour must lay on the strip going from the w-edge of $I_{i\ell}$ to the rear of the vehicle. The dashed strip above item I_{il} in Figure 7.2 (a) shows the forbidden loading area.

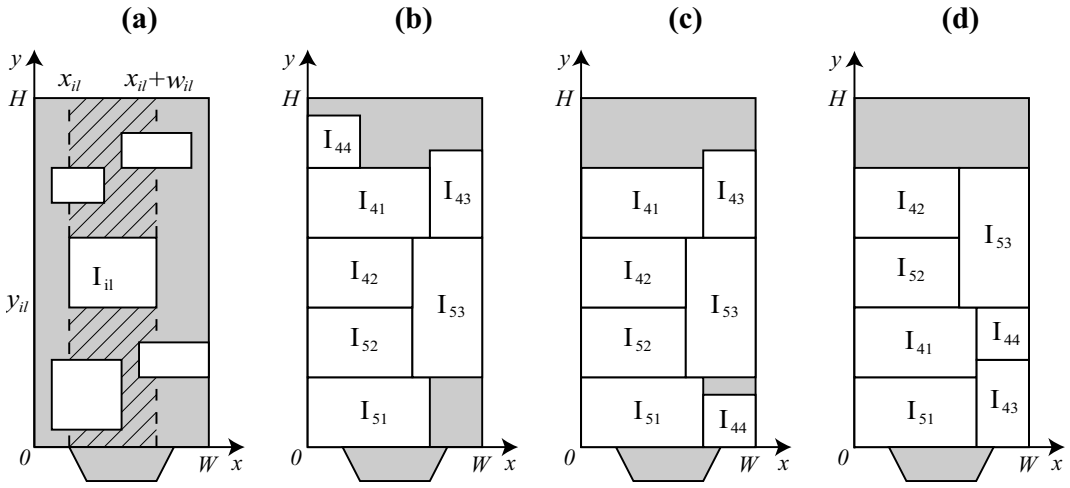

Figure 7.2: *The loading surface.*
(a) *forbidden areas*; (b) *sequential loading*; (c), (d) *unrestricted loadings.*

This constraint can arise in practice when the items have great weight, size or fragility, so that moving them inside the vehicle is extremely difficult or impossible. The sequential variant is the one studied in [98]. In Figure 7.1, we show a simple example with 3 vehicles of weight capacity $D = 100$, 8 customers and 21 items. The three patterns given for the vehicles are feasible sequential loadings for the freight delivery. (An example of non-sequential loading is given in Figure 7.2.)

We assume in the following that all input data are positive integers. The loading area of a vehicle is represented in the positive quadrant of a Cartesian coordinate system, with its bottom left corner in $(0, 0)$ and the W-edge (resp. H-edge) parallel to the x-axis (resp. y-axis), and the vehicle rear coinciding with segment $[(0, H), (W, H)]$. The position of an item $I_{i\ell}$ in a loading pattern is given by the coordinates of its bottom left point, $(x_{i\ell}, y_{i\ell})$. The coordinate system is shown in Figure 7.2: by assuming that $W = 100$ and $H = 200$, in Figure 7.2 (b) the position of item I_{51} is $(0, 0)$, that of item I_{52} is $(0, 40)$, and so on.

7.3 A Tabu Search Approach

Tabu Search is a very effective and simple technique for the solution of optimization problems. It was first proposed by Glover [65, 66] and has since been applied to a large number of different problems (see, e.g., Glover [67] and Glover and Laguna [71] for surveys).

Tabu Search starts from an initial (usually feasible) solution and iteratively moves to a new one selected in a certain *neighborhood* of the current solution. At each iteration, the *move* yielding the best solution in the neighborhood is selected, even if this results in a worse solution. To avoid cycling, solutions possessing some attributes of previously visited solutions are declared tabu for a number of iterations. In order to obtain good practical results, this simple scheme is usually enriched with additional features. An *intensification* phase is often performed to accentuate the search in a promising area, while *diversification* is necessary to escape from a poor or already extensively searched area.

The Tabu Search approach introduced in the next sections can accept moves producing infeasible tours in the following sense. For the sequential 2L-CVRP, all tours considered in the search satisfy the sequence constraint. However, both in the sequential and the unrestricted 2L-CVRP, the tours can be *weight-infeasible* if the total weight exceeds D, or *load-infeasible* if the packing needs a loading surface of height exceeding H. The algorithm never produces packings requiring a loading surface wider than W or implying overlapping items. Infeasible moves are assigned a *penalty* proportional to the level of the violation.

Thus, when performing a move, the algorithm must consider the improvement of the current solution in terms of total edge cost, and check the feasibility of the candidate heuristic tour. For the routing aspects, we have adopted an approach (described later in Section 7.3.3) derived from some successful features of Taburoute, a Tabu Search heuristic developed by Gendreau, Hertz and Laporte [58] for the solution of vehicle routing problems. For what concerns feasibility, the weight constraint is immediate to check, but the loading constraint requires the solution of an NP-hard problem. We have thus developed a heuristic algorithm, described in the next section, executed for each set of clients assigned to a vehicle: it outputs a two-dimensional pattern of width W and height $Hstrip$, to be tested against the available height H.

7.3.1 A heuristic algorithm to check the loading constraints

Let $I(k)$ (see (7.1)) be the set of items currently assigned to vehicle k. In this subsection we describe two heuristics, $\mathrm{LH_{2SL}}$ and $\mathrm{LH_{2UL}}$, to check the loading constraints for vehicle k in the sequential and the unrestricted case, respectively. The algorithms iteratively apply an inner procedure based on the heuristic rule used in algorithm $\mathrm{TP_{RF}}$ (*Touching Perimeter*) proposed by Lodi, Martello and Vigo [112] for the two-dimensional bin packing problem, and applied by Iori, Martello and Monaci [97] to the two-dimensional strip packing problem. We have developed two inner procedures, $\mathrm{TP_{2SL}}$ and $\mathrm{TP_{2UL}}$, for the sequential and the unrestricted case, respectively. Both procedures assign one item at a time (according to a given item sorting) to a strip of width W, trying to maximize at each iteration the percentage of the item perimeter touching the strip or other items already packed (*touching perimeter*). Each item is packed in the *normal position* (as defined by Christofides and Whitlock [25]), i.e., with its bottom edge touching either the bottom of the strip or the top edge of another item, and with its left edge touching either the left edge of the strip or the right edge of another item.

We next provide the description of procedure $\mathrm{TP_{2SL}}$ (sequential case), shown in Figure 7.3. The first call to $\mathrm{TP_{2SL}}$ is performed by $\mathrm{LH_{2SL}}$ with an item sorting obtained by scanning the clients assigned to a route in reverse order of visit and, for each client, by listing the items in the corresponding lot according to non-increasing width, breaking ties by non-increasing height. Subsequent calls are performed by perturbing this order, as will be shown. Procedure $\mathrm{TP_{2SL}}$ can terminate in one of three possible ways: with a feasible packing, with a load-infeasible packing on a surface $W \times Hstrip$ with $Hstrip > H$ (to be penalized within Tabu Search), or with no packing at all. Since the last outcome cannot occur at the first call, procedure $\mathrm{LH_{2SL}}$ always returns a feasible or load-infeasible packing.

The procedure works as follows. The first item is packed in position $(0, 0)$. The next item, say $I_{i\ell}$, is packed, if possible, with its bottom-left corner at a coordinate (x, y) that maximizes the touching perimeter, provided that (i) the unloading sequence is feasible (see Figure 7.2 (a)), (ii) the induced strip height does not exceed the vehicle limit. If no such coordinate exists, the item is packed in that position, among those satisfying (i) (if any), for which the induced strip height is smaller. With item sortings other than the initial one it is possible that an item is encountered for which no packing satisfying (i) exists, in which case a dummy infinite height is returned. Figure 7.2 (b) shows the packing produced by $\mathrm{TP_{2SL}}$ for the second vehicle of Figure 7.1 with the initial sorting.

Let $\overline{m}$ be the total number of items required by the clients of the considered route. For each item, the number of candidate normal packing positions (set P) is $O(\overline{m})$. The feasibility check of Step 2.1 (c) and the touching perimeter computation of step 2.3 require $O(\overline{m})$ time. The overall time complexity of $\mathrm{TP_{2SL}}$ is thus $O(\overline{m}^3)$.

procedure $\mathbf{TP}_{2SL}(I(k))$
1. pack the first item of $I(k)$ in position $(0,0)$ and set $\overline{H}$ at its height;
2. **for each** successive item, say I of size (w,h), of $I(k)$ **do**
   2.1 P:= {set of normal packing positions (x,y) : (a), (b), (c) below hold}
   (a) $x+w \leq W$;
   (b) the rectangle of sides $([(x,y),(x+w,y)], [(x,y),(x,y+h)])$ is free;
   (c) for each already packed item $\overline{I}$, say of size $(\overline{w},\overline{h})$ packed at $(\overline{x},\overline{y})$,
   (c.1) if $\overline{I}$ belongs to the lot of a previous client then
   either $(\overline{y} \geq y)$ or $(\overline{x}+\overline{w} \leq x)$ or $(\overline{x} \geq x+w)$;
   (c.2) if $\overline{I}$ belongs to the lot of a successive client then
   either $(\overline{y} \leq y+h)$ or $(\overline{x}+\overline{w} \leq x)$ or $(\overline{x} \geq x+w)$;
   2.2 **if** $P=\emptyset$ **then return** ∞ **else** $\overline{P}=\{(x,y)\in P : y+h \leq H\}$;
   2.3 **if** $\overline{P} \neq \emptyset$ **then**
   pack I in the position $(x,y)\in\overline{P}$ producing maximum touching perimeter
   else pack I in the position $(x,y)\in P$ for which $y+h$ is a minimum;
   2.4 $\overline{H} := \max\{\overline{H}, y+h\}$
   end for;
3. **return** $\overline{H}$.

Figure 7.3: *Inner heuristic for producing sequential loadings.*

Procedure TP_{2SL} is iteratively invoked by algorithm LH_{2SL} on modified item sequences as shown in Figure 7.4, where λ is the maximum number of times the inner procedure is executed. On input, the items are ordered according to the initial sorting previously described. The output is either a feasible packing if the returned value is $Hstrip \leq H$, or a load-infeasible packing if $Hstrip > H$. Since Procedure TP_{2SL} finds one of these two kinds of packing when invoked with the initial sorting, LH_{2SL} will always return a packing.

Procedure $\mathbf{LH}_{2SL}(I(k),\lambda)$
1. $Hstrip := \mathrm{TP}_{2SL}(I(k))$, $it := 1$;
2. **while** $Hstrip > H$ **and** $it \leq \lambda$ **do**
   2.1 randomly select two items and switch their positions;
   2.2 $\xi := \mathrm{TP}_{2SL}(I(k))$;
   2.3 $Hstrip := \min\{Hstrip, \xi\}$;
   2.4 $it := it+1$;
   end while
3. **return** $Hstrip$.

Figure 7.4: *Heuristic algorithm for checking the sequential loading constraints.*

For the unrestricted case, the corresponding procedure, LH_{2UL}, is identical to LH_{2SL} except for the call to a procedure TP_{2UL} which is obtained from TP_{2SL} by dropping condition (c) of Step 2.1. (Note that in this case P will never be empty at Step 2.2.) For the vehicle of Figure 7.2 (b), Figures 7.2 (c) and (d) show two packings produced by TP_{2UL}: the former with the initial sorting, the latter with items sorted by decreasing width, breaking ties by decreasing height.

7.3.2 Initial solution

We use two heuristic algorithms (each in two versions, for the sequential and the unrestricted cases) for determining an initial solution. The first algorithm, IHG_{2SL} or IHG_{2UL}, works for general 2L-CVRP

instances and is always executed. For the special case of Euclidean instances, a second heuristic algorithm, $\text{IHE}_{2\text{SL}}$ or $\text{IHE}_{2\text{UL}}$, is also executed.

Algorithm $\text{IHG}_{2\text{SL}}$ (resp. $\text{IHG}_{2\text{UL}}$) was obtained by embedding the loading and sequencing constraints (resp. the loading constraints) in the classical Clarke and Wright [26] savings algorithm. Whenever an attempt is made to merge two routes, we check whether the weight constraint is satisfied, and, if so, we execute procedure $\text{LH}_{2\text{SL}}$ (resp. $\text{LH}_{2\text{UL}}$) of Section 7.3.1 to check the load feasibility. The algorithm is first executed by only accepting merges for which a feasible packing is returned. If in this way we obtain a solution that uses v vehicles, the execution terminates. Otherwise, starting from the current solution, we execute a new round where load-infeasible packings are accepted.

The Euclidean algorithm, $\text{IHE}_{2\text{SL}}$ (resp. $\text{IHE}_{2\text{UL}}$), was obtained by embedding the loading and sequencing constraints (resp. the loading constraints) in the algorithm developed by Cordeau, Gendreau and Laporte [33] for periodic and multi-depot VRPs. We start by randomly selecting a radius emanating from the depot, and by sorting the clients by increasing value of the angle between the radius and the segment connecting the client to the depot. The first client is assigned to the first vehicle. We then attempt to assign each subsequent client to the current vehicle by checking weight capacity and loading feasiblity through $\text{LH}_{2\text{SL}}$ (resp. $\text{LH}_{2\text{UL}}$). For the first $v-1$ vehicles, only feasible packings are accepted, while a load-infeasible packing is only accepted for the last vehicle. The best feasible solution determined in the initialization phase (if any) is stored as the incumbent solution.

7.3.3 Tabu Search

As previously observed, our Tabu Search heuristic has to deal with weight-infeasible or load-infeasible solutions. We treat infeasibilities as penalties in an objective function to be minimized. Let s denote a solution, consisting of a set of $\widetilde{v}$ routes, with $1 \leq \widetilde{v} \leq v$, in which each client belongs to exactly one route. Let $c(k)$ be the total cost of the edges in route k. As in Taburoute, we express the objective function as a sum of three terms:

$$z'(s) = z(s) + \alpha q(s) + \beta h(s) \tag{7.2}$$

with α and β given positive parameters. In our case, $q(s)$ and $h(s)$ represent the entity of the violation, if any, of the weight and loading constraints, respectively. Using the notation introduced in Section 7.2 and denoting by H_k the height of the two-dimensional loading of vehicle k, we have

$$z(s) = \sum_{k=1}^{\widetilde{v}} c(k) \tag{7.3}$$

$$q(s) = \sum_{k=1}^{\widetilde{v}} \left[\sum_{i \in S(k)} d_i - D \right]^+ \tag{7.4}$$

$$h(s) = \sum_{k=1}^{\widetilde{v}} [H_k - H]^+ \tag{7.5}$$

where $[a]^+ = \max\{0, a\}$.

A move consists of selecting a client i, currently inserted in a route k, removing it from the route and inserting it into a new route k'. The cost of the new solution is obtained through the GENI (Generalized Insertion Procedure) heuristic developed by Gendreau, Hertz and Laporte [59] for the Travelling Salesman Problem. GENI considers a subset of the possible 4-opt modifications of the two tours (the one from which the client is deleted and the one where it is inserted), selecting the best one for each tour. Let $N_p(i)$ (neighborhood of client i) be the set of the p vertices closest to i. In order to reduce the number of possible moves, only empty routes and routes containing at least one neighbor of i are considered for possible insertion of i. A good compromise between computing time and solution quality was experimentally obtained with $p = 10$.

At each iteration, all possible moves are evaluated, i.e., for each client we consider its possible insertion into all routes satisfying the neighbor condition above: for each move, the resulting cost $z'(s)$ (see (7.2)–(7.5)) is obtained by determining $z(s)$ through GENI, directly computing $q(s)$, and computing $h(s)$ through procedure $\mathrm{LH}_{2\mathrm{SL}}$ (or $\mathrm{LH}_{2\mathrm{UL}}$). In certain situations, a *penalty factor* π (see below, equation (7.6)) is possibly added to the cost in order to diversify the search. The move producing the solution with minimum overall cost is selected. After a move has been performed by removing client i from route k, reinserting i in k is declared tabu for the next ϑ iterations. An aspiration criterion is based on an $n \times v$ array M that stores in M_{ij} the value of the best feasible solution so far obtained, if any, with client i assigned to route j ($M_{ij} = \infty$ if no such solution has been identified). A tabu move that reinserts client i in route k can be accepted if it improves the current M_{ij} value. The value of ϑ was experimentally determined as $\vartheta = \log(nv)$.

Three main parameters govern the search process: α controls weight constraint violations (see (7.2)), while β and λ control packing constraint violations (see (7.2) and Section 7.3.1). In the initialization phase α and β are set to 1, and λ to 2. In the iterative phase the three parameters are recursively updated following the characteristics of the solutions produced by the moves. If the current solution is (resp. is not) weight-infeasible we set $\alpha = \alpha(1+\delta)$ (resp. $\alpha = \alpha/(1+\delta)$). In addition, if the current solution is (resp. is not) load-infeasible we set $\beta = \beta(1+\delta)$ and $\lambda = \min\{\lambda+1, 4\}$ (resp. $\beta = \beta/(1+\delta)$ and $\lambda = \max\{\lambda-1, 1\}$. A good value of δ was experimentally determined as $\delta = 0.01$.

We use a simple diversification technique to escape from poor local minima. Let s_a be the current solution, and s_b the solution produced by a move that removes customer i from route k. As in Taburoute, if $z'(s_b) \geq z'(s_a)$, we sometimes add to $z'(s_b)$ a penalty factor π, proportional to the frequency of the considered move during the previous iterations. Let ρ_{ik} denote the number of times customer i was removed from route k, divided by the total number of iterations. The penalty factor is then

$$\pi = \rho_{ik}(z'(s_b) - z'(s_a))\gamma, \tag{7.6}$$

with γ set to $\sqrt{nv}$, as suggested by Taillard [143] and Cordeau, Laporte and Mercier [32]. In this way, when no downhill move is found, the algorithm is more likely to escape from poor local search areas. In order to deeply explore promising areas of the solution space, we extensively use intensification algorithms. Because of their importance they are described in the next section.

7.4 Intensification

Two kinds of intensification techniques are adopted, one executed periodically, every ω moves, and one (computationally heavier) executed whenever a weigth-feasible potential new incumbent is obtained. Both intensification processes make use of a procedure ($\mathrm{Repack}_{2\mathrm{SL}}(k)$ for the sequential case, or $\mathrm{Repack}_{2\mathrm{UL}}(k)$ for the unrestricted case) that looks for a feasible packing for the items of each route k of a load-infeasible solution s, i.e., one for which $H_k > H$.

Recall that $I(k)$ is the set of items associated with the clients of route k. In Figure 7.5 Procedure $\mathrm{Repack}_{2\mathrm{SL}}(I(k), \lambda)$ is shown. It starts by computing lower bounds (Martello and Vigo [120]) for the two-dimensional bin packing problem instance induced by the items associated with route k and bin size $W \times H$. If more than one bin is needed, no feasible loading exists and the procedure terminates. Otherwise we execute the heuristic search $\mathrm{LH}_{2\mathrm{SL}}$ of Section 7.3.1. For this computation, it turned out to be convenient to allow a much higher number of calls, $\lambda = 10$, to the inner heuristic $\mathrm{TP}_{2\mathrm{SL}}$. If no feasible packing is still found, we execute the branch-and-bound algorithm of Iori, Salazar González and Vigo [98] with a limit of 1 000 backtrackings.

Procedure $\mathrm{Repack}_{2\mathrm{UL}}$ is identical to $\mathrm{Repack}_{2\mathrm{SL}}$ except for the call to $\mathrm{LH}_{2\mathrm{UL}}$ instead of $\mathrm{LH}_{2\mathrm{SL}}$, and for the execution of a truncated branch-and-bound search that does not consider the sequence constraints.

Procedure Repack$_{2SL}(I(k), \lambda)$
1. $L :=$ lower bound for the 2BPP instance defined by $I(k)$ with bin size $W \times H$;
2. **if** $L > 1$ **then return** ∞;
3. $\xi := \mathrm{LH}_{2SL}(I(k), \lambda)$;
4. **if** $\xi \leq H$ **then return** ξ;
5. execute a truncated branch-and-bound search;
6. **if** a solution of value 1 is found **then return** H
   else return ∞.

Figure 7.5: *Intensification for packing constraints.*

7.4.1 Periodic Intensification

Every ω moves, a local search for improving the incumbent solution is performed through a modified version of the heuristic algorithm of Iori, Salazar González and Vigo [98]. This algorithm is embedded within a branch-and-cut approach and operates on a modified cost matrix $\overline{c}$ obtained by setting, for each edge (i, j), $\overline{c}_{ij} = c_{ij}(1 - x_{ij}/2)\mu$, where x_{ij} is the (possibly fractional) value of the 0-1 decision variable associated with the edge and μ is a randomization factor uniformly distributed in [0,1]. The new solution is then obtained by iteratively building routes through a parametric version of the savings algorithm. Each route (client set) S is constructed by first selecting the client i for which the quantity $\sigma_c(c_{0i} + c_{i0}) + \sigma_d d_i + \sigma_a a_i$ is a maximum (σ_c, σ_d and σ_a prefixed parameters), and then iteratively inserting in S the client j for which the residual weight capacity $d_{res} = D - \sum_{i \in S} d_i - d_j$ and loading area $a_{res} = A - \sum_{i \in S} a_i - a_j$ are non-negative, and $\sigma_c($ insertion cost $) + \sigma_d d_{res} + \sigma_a a_{res}$ is a minimum. Our implementation, procedure Improve$_{2SL}$, is shown in Figure 7.6.

Procedure Improve$_{2SL}$
1. create the modified cost matrix $\overline{c}$ and set $P := \emptyset$;
2. **for each** triplet $\sigma = (\sigma_c, \sigma_d, \sigma_a) \in T$ **do**
   $s :=$ SAVINGS($\widetilde{c}$, σ);
   if s includes no more than v routes **then**
   if $z(s) < z$ **then** $P := P \cup \{s\}$;
   $z_{\mathrm{old}} := \infty$;
   while $z(s) < z_{\mathrm{old}}$ **do**
   $z_{\mathrm{old}} := z(s)$;
   $s :=$ TWO-OPT(c, s);
   if $z(s) < z$ **then** $P := P \cup \{s\}$
   end while
   end if
   end for.
3. sort the solutions $s \in P$ by non-decreasing $z(s)$ value;
4. **for each** $s \in P$ in order **do**
   for each route k of s **do**
   $\xi := \mathrm{Repack}_{2SL}(I(k), \lambda)$;
   if $\xi > H$ **then break**
   end for;
   if $\xi \leq H$ **then return** s
   end for;
   return $\emptyset$.

Figure 7.6: *Algorithm for improving the incumbent solution.*

The modified cost matrix is obtained by setting

$$\overline{c}_{ij} = \begin{cases} rc_{ij} \text{ (with } r \text{ uniformly random in [0,2])} & \text{if } (i,j) \text{ belongs to a route,} \\ c_{ij} & \text{otherwise.} \end{cases} \tag{7.7}$$

and only clients for which the residual loading area a_{res} is no less than a threshold value τ are considered for possible insertion, in order to take into account the difficulty of packing the corresponding items. The value τ is randomly generated in $[0, A \cdot R/4]$, where

$$R = \max\{\max_{i\ell}\{h_{i\ell}/H\}, \max_{i\ell}\{w_{i\ell}/W\}\} \tag{7.8}$$

(maximum relative size of an item) measures the "difficulty" of the packing problem associated with the instance. In other words, we decrease the residual packing area by a percentage that takes into account the probability that portions of the loading surface remain unused.

We use a set of five triplets for the values of $(\sigma_c, \sigma_d, \sigma_a)$: $T = \{(1,0,0), (\frac{1}{2}, \frac{1}{2}, 0), (\frac{1}{2}, 0, \frac{1}{2}), (0, \frac{1}{2}, \frac{1}{2}), (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})\}$. The savings algorithm is executed for each triplet, and all solutions that use no more than v routes are improved through 2-opt modifications. In addition, all solutions of value less than that of the incumbent are stored in a *pool* P. We then consider the solutions in P in increasing value. For each solution, we try to obtain a feasible packing for each route through procedure $\text{Repack}_{2\text{SL}}(I(k), \lambda)$, stopping as soon as an overall feasible solution (if any) is obtained. Given $\overline{c}$ and a triplet σ, let $s = \text{SAVINGS}(\overline{c}, \sigma)$ be the solution built by the savings algorithm, and TWO-OPT(c,s) the improved solution produced by the first improvement (if any) found through the two-opt local search algorithm (TWO-OPT$(c, s) \equiv s$ if no improvement was found). All solution values $z(s)$ are obviously evaluated using the original cost matrix c.

Procedure $\text{Improve}_{2\text{UL}}$ is identical to $\text{Improve}_{2\text{SL}}$ except for the call to $\text{Repack}_{2\text{UL}}(I(k), \lambda)$ instead of $\text{Repack}_{2\text{SL}}(I(k), \lambda)$. Basing on the outcome of computational testings, we set ω to 1 for $n \leq 40$ and to 5 for larger values.

7.4.2 Special intensification

Whenever a solution s having a cost $z(s)$ lower than that of the incumbent is produced, different actions are taken, depending on the feasibility of the weight and of the loading associated with s:

(i) if s is weight-infeasible, no attempt is made to make it feasible;

(ii) if s is totally feasible, the incumbent is updated, and some more time is spent in an attempt to further improve s on an enlarged neighborhood. This is achieved by doubling, for each client i, the size of $N_p(i)$, and applying GENI for each resulting possible move. In our implementation this results in considering all routes that are empty or contain at least one of the 20 vertices closest to i;

(iii) if s is weight-feasible but load-infeasible, we first iteratively execute procedure Repack for each route k of s for which $H_k > H$. If a feasible solution is obtained in this way, the additional intensification of case (ii) above is further performed.

7.5 Computational Results

The Tabu Search heuristic developed in the previous sections was coded in C and tested on a test set obtained by modifying instances from the literature. All experiments were executed on a Pentium IV 1700 MHz. The graph and the weights demanded by the customers were obtained from CVRP instances (test instances for CVRP are described in [144], and can be downloaded from `http://www.or.deis.unibo.it/research.html`). The number of items and their dimensions were created according to the five classes introduced in [98] and in IV, namely:

Class 1: each customer i, for $i = 1, \ldots, n$ is assigned one item with unit width and height.

Classes 2 – 5: a random uniform distribution on a certain interval (see Table 7.1, column 2) is used to generate the number m_i of items for each customer i. Each item is then randomly assigned, with equal probability, one out of three possible shapes (*vertical*, *homogeneous* or *horizontal*). Finally, the dimensions of the items are uniformly randomly generated in a given interval (see again Table 7.1, columns 3–8).

The values $H = 40$ and $W = 20$ were chosen for the dimensions of the loading area. Class 1 corresponds to pure CVRP instances, for which no loading constraint is imposed. The other classes proved to be a challenging mix of items of different sizes.

Table 7.1: Classes 2 –5

		Vertical		*Homogeneous*		*Horizontal*	
Class	m_i	$h_{i\ell}$	$w_{i\ell}$	$h_{i\ell}$	$w_{i\ell}$	$h_{i\ell}$	$w_{i\ell}$
2	$[1,2]$	$[\frac{4H}{10}, \frac{9H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{2H}{10}, \frac{5H}{10}]$	$[\frac{2W}{10}, \frac{5W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{4W}{10}, \frac{9W}{10}]$
3	$[1,3]$	$[\frac{3H}{10}, \frac{8H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{2H}{10}, \frac{4H}{10}]$	$[\frac{2W}{10}, \frac{4W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{3W}{10}, \frac{8W}{10}]$
4	$[1,4]$	$[\frac{2H}{10}, \frac{7H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{H}{10}, \frac{4H}{10}]$	$[\frac{W}{10}, \frac{4W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{2W}{10}, \frac{7W}{10}]$
5	$[1,5]$	$[\frac{H}{10}, \frac{6H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{H}{10}, \frac{3H}{10}]$	$[\frac{W}{10}, \frac{3W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{W}{10}, \frac{6W}{10}]$

Some details on the instances generated in this way are reported in Table 7.2. For each CVRP instance, we created five 2L-CVRP instances (one per class) by generating the number of vehicles v through heuristic $\mathrm{LH}_{\mathrm{2SL}}$ of Section 7.3.1, modified so as to pack the items into finite $W \times H$ bins instead of a strip. The CVRP instances 1–16 were already used in [98] and IV, producing 80 different 2L-CVRP instances according to the five above classes, while the 100 instances obtained from 17–36 are considered here for the first time. In the lines of Table 7.2 (one for each group of five instances), the first column gives the name of the original CVRP instance and the second one the number of customers (n). The five following groups of three columns give, for each class, the total number of items ($M = \sum_{i=1}^{n} m_i$), the number of vehicles (v) and a lower bound (LB) on the minimum number of vehicles needed to ensure feasibility. This lower bound, reported here in order to show that the produced instances are indeed "reasonable", was computed as the maximum among two valid lower bounds: the number of vehicles in the original CVRP instance and the value obtained by solving the two-dimensional bin packing problem induced by the item sizes and the vehicle loading area. This second part of the bound was computed through the code by Martello, Pisinger and Vigo [116], available at `http://www.diku.dk/~pisinger`. (This code works for the three-dimensional bin packing problem and can obviously be used for the two-dimensional case as well.) The table shows that the v and LB values are very close: out of a total of 180 instances, these values are identical in 73 cases, and differ by at most two units in 57 cases. The complete set of 180 test instances that were obtained may be downloaded from `http://www.or.deis.unibo.it/research.html`.

In tables 7.3 and 7.4, the performance of our approach is compared with the branch-and-cut algorithm by Iori, Salazar González and Vigo [98] on small-size instances involving less than 45 customers. Since the latter approach does not consider single-customer routes and always uses the v available vehicles, the Tabu Search algorithm was executed by making infeasible (through penalty factors) any solution not satisfying these properties. In addition, as in [98], the edge costs were computed by rounding down to the next integer the Euclidean distances between vertex pairs. The branch-and-cut algorithm had a time limit of 24 hours per instance, as in the experiments presented in [98]. A convenient policy for controlling the Tabu Search was experimentally determined as follows. The algorithm is halted after $2n^2v$ iterations, or after one hour of CPU time.

For each instance, tables 7.3 and 7.4 give, the best solution obtained both by branch-and-cut (z_{bc}) and by Tabu Search (z_{TS}), the elapsed CPU time in seconds when the best solution was found

Table 7.2: Details on the instances. Original CVRP instance, number of customers; for each class, number of items, number of vehicles, lower bound on a feasible number of vehicles.

		Class 1			*Class 2*			*Class 3*			*Class 4*			*Class 5*		
Instance	n	M	v	LB	M	v	LB	M	v	LB	M	v	LB	M	v	LB
1) E016-03m	15	15	3	3	24	3	3	31	3	3	37	4	3	45	4	3
2) E016-05m	15	15	5	5	25	5	5	31	5	5	40	5	5	48	5	5
3) E021-04m	20	20	4	4	29	5	4	46	5	4	44	5	4	49	5	4
4) E021-06m	20	20	6	6	32	6	6	43	6	6	50	6	6	62	6	6
5) E022-04g	21	21	4	4	31	4	4	37	4	4	41	4	4	57	5	4
6) E022-06m	21	21	6	6	33	6	6	40	6	6	57	6	6	56	6	6
7) E023-03g	22	22	3	3	32	5	4	41	5	4	51	5	4	55	6	3
8) E023-05s	22	22	5	5	29	5	5	42	5	5	48	5	5	52	6	5
9) E026-08m	25	25	8	8	40	8	8	61	8	8	63	8	8	91	8	8
10) E030-03g	29	29	3	3	43	6	5	49	6	4	72	7	6	86	7	5
11) E030-04s	29	29	4	4	43	6	5	62	7	6	74	7	6	87	7	5
12) E031-09h	30	30	9	9	50	9	9	56	9	9	82	9	9	101	9	9
13) E033-03n	32	32	3	3	44	7	5	56	7	5	78	7	6	102	8	5
14) E033-04g	32	32	4	4	44	7	5	57	7	5	65	7	5	87	8	4
15) E033-05s	32	32	5	5	50	6	5	59	6	6	90	8	7	114	8	6
16) E036-11h	35	35	11	11	56	11	11	74	11	11	93	11	11	114	11	11
17) E041-14h	40	40	14	14	60	14	14	73	14	14	96	14	14	127	14	14
18) E045-04f	44	44	4	4	66	9	7	87	10	8	112	10	8	122	10	6
19) E051-05e	50	50	5	5	82	11	9	103	11	10	134	12	10	157	12	8
20) E072-04f	71	71	4	4	104	14	12	151	15	13	178	16	13	226	16	13
21) E076-07s	75	75	7	7	114	14	12	164	17	14	168	17	14	202	17	14
22) E076-08s	75	75	8	8	112	15	13	154	16	14	198	17	14	236	17	14
23) E076-10e	75	75	10	10	112	14	13	155	16	14	179	16	14	225	16	14
24) E076-14s	75	75	14	14	124	17	14	152	17	14	195	17	14	215	17	14
25) E101-08e	100	100	8	8	157	21	18	212	21	18	254	22	19	311	22	19
26) E101-10c	100	100	10	10	147	19	16	198	20	17	247	20	18	310	20	18
27) E101-14s	100	100	14	14	157	19	17	211	22	19	245	22	19	320	22	19
28) E121-07c	120	120	7	7	183	23	20	242	25	21	299	25	21	384	25	21
29) E135-07f	134	134	7	7	197	24	21	262	26	22	342	28	24	422	28	24
30) E151-12b	150	150	12	12	225	29	25	298	30	27	366	30	27	433	30	27
31) E200-16b	199	199	16	16	307	38	33	402	40	35	513	42	37	602	42	37
32) E200-17b	199	199	17	17	299	38	33	404	39	34	497	39	34	589	39	34
33) E200-17c	199	199	17	17	301	37	32	407	41	35	499	41	36	577	41	36
34) E241-22k	240	240	22	22	370	46	40	490	49	42	604	50	44	720	50	44
35) E253-27k	252	252	27	27	367	45	39	507	50	43	634	50	45	762	50	45
36) E256-14k	255	255	14	14	387	47	41	511	51	44	606	51	44	786	51	44

Table 7.3: Performance of the Tabu Search heuristic with respect to branch-and-cut. Sequential instances, integer edge costs.

Instance		*Branch-and-cut*				*Tabu Search*			
no	*cl*	z_{bc}		$seconds_h$	$seconds_{tot}$	z_{TS}	$seconds_h$	$seconds_{tot}$	$\%gap$
1	1	273	*	3.80	3.86	273	0.05	2.45	0.00
	2	285	*	51.38	68.81	285	0.19	4.56	0.00
	3	280	*	17.06	21.42	280	1.33	7.84	0.00
	4	288	*	1.80	2.36	290	0.30	7.53	0.69
	5	279	*	53.08	53.08	279	2.28	15.75	0.00
2	1	329	*	0.19	0.55	329	0.09	1.36	0.00
	2	342	*	11.08	11.95	342	1.12	1.98	0.00
	3	347	*	6.39	8.14	350	0.17	3.17	0.86
	4	336	*	22.42	23.42	336	0.06	6.70	0.00
	5	329	*	29.22	29.36	329	0.34	6.08	0.00
3	1	351	*	14.92	15.61	351	0.25	8.36	0.00
	2	389	*	32.59	65.78	407	8.95	9.77	4.63
	3	387	*	6.03	6.06	387	0.97	19.95	0.00
	4	374	*	39.28	39.36	374	14.28	21.94	0.00
	5	369	*	0.17	0.22	369	0.95	29.88	0.00
4	1	423	*	0.38	0.41	423	0.16	5.66	0.00
	2	434	*	5.30	5.47	434	0.27	7.41	0.00
	3	432	*	5.23	8.16	438	5.03	12.77	1.39
	4	438	*	23.95	26.38	451	0.81	16.91	2.97
	5	423	*	23.00	44.70	423	1.16	27.13	0.00
5	1	367	*	0.08	0.08	367	2.94	12.83	0.00
	2	380	*	7.98	8.36	396	4.00	19.69	4.21
	3	373	*	1.81	1.91	377	0.58	20.55	1.07
	4	377	*	50.20	50.30	406	5.92	33.00	7.69
	5	389	*	2928.20	2928.27	389	5.73	57.84	0.00
6	1	488	*	5.89	10.69	488	0.13	10.30	0.00
	2	491	*	145.83	145.86	498	1.84	13.98	1.43
	3	496	*	135.41	150.37	496	11.25	16.95	0.00
	4	489	*	14.12	16.56	503	0.14	40.17	2.86
	5	488	*	10.76	13.34	488	0.70	34.19	0.00
7	1	558	*	0.02	0.02	558	0.31	22.02	0.00
	2	724	*	32.05	32.47	752	0.67	18.89	3.87
	3	698	*	3.19	4.39	704	19.86	29.78	0.86
	4	714	*	2596.83	2597.27	742	26.55	50.36	3.92
	5	742	*	738.94	747.17	743	16.06	75.08	0.13
8	1	657	*	0.03	0.05	657	6.99	31.33	0.00
	2	720	*	75.94	91.81	720	3.06	20.09	0.00
	3	730	*	70.03	72.95	752	20.55	33.38	3.01
	4	701	*	7.44	14.12	722	11.22	49.95	3.00
	5	721	*	1128.86	1128.86	736	12.05	90.17	2.08
9	1	609	*	6.17	31.91	609	1.89	11.87	0.00
	2	612	*	453.52	460.52	612	2.89	15.44	0.00
	3	615	*	164.83	194.31	626	7.89	38.45	1.79
	4	626	*	852.14	1593.34	627	5.87	43.47	0.16
	5	609	*	47.39	69.17	609	8.47	81.86	0.00

Table 7.4: Performance of the Tabu Search heuristic with respect to branch-and-cut. Sequential instances, integer edge costs.

Instance		*Branch-and-cut*				*Tabu Search*			
no	*cl*	z_{bc}		$seconds_h$	$seconds_{tot}$	z_{TS}	$seconds_h$	$seconds_{tot}$	$\%gap$
10	1	524		1226.17	86400.52	544	19.91	97.33	3.82
	2	774		48373.34	86401.15	703	3.81	72.20	−9.17
	3	638	*	2304.46	3149.99	676	47.86	118.67	5.96
	4	738	*	12671.31	12696.29	773	47.28	156.94	4.74
	5	706		48220.23	70308.02	724	84.47	308.87	2.55
11	1	500	*	0.13	0.14	500	0.81	107.81	0.00
	2	789		99.80	86400.46	734	4.69	72.23	−6.97
	3	763		63747.77	86400.32	785	72.17	101.69	2.88
	4	881		85181.52	86400.83	877	196.44	209.52	−0.45
	5	695		64392.94	86400.06	696	279.17	387.00	0.14
12	1	599		50093.46	86400.34	598	19.41	33.80	−0.17
	2	625		75.60	86400.67	628	9.64	42.87	0.48
	3	597		36171.03	86400.58	597	12.91	50.75	0.00
	4	624		86321.63	86400.16	640	95.98	120.59	2.56
	5	602		22352.36	86400.36	597	103.47	188.24	−0.83
13	1	1991	*	13.18	14.52	1991	47.36	218.69	0.00
	2	3523		10784.11	86400.30	2775	43.75	123.06	−21.23
	3	2570	*	3087.87	32124.54	2696	121.50	170.34	4.90
	4	2673		34999.17	86400.38	2743	29.31	277.05	2.62
	5	2807		83628.40	86400.10	2737	237.52	691.88	−2.49
14	1	827		25734.79	86400.43	823	21.76	145.03	−0.48
	2	1459		80521.15	86400.84	1266	52.33	144.38	−13.23
	3	1211		42204.75	86400.09	1204	137.94	207.09	−0.58
	4	1166	*	25391.47	25542.61	1187	108.52	324.94	1.80
	5	1504		80511.08	86400.08	1309	103.17	895.02	−12.97
15	1	907	*	17.81	17.84	907	51.92	196.37	0.00
	2	1203		386.89	86401.44	1135	94.66	133.89	−5.65
	3	1405		15880.73	86401.21	1183	37.11	205.87	−15.80
	4	1358		55614.54	86400.27	1372	268.25	332.20	1.03
	5	1390		59867.80	86400.10	1361	651.45	671.70	−2.09
16	1	682	*	7086.48	7086.71	682	56.06	96.58	0.00
	2	682	*	1766.95	3374.76	682	20.55	91.70	0.00
	3	682	*	345.15	3853.35	682	15.33	138.09	0.00
	4	691	*	33375.93	33391.84	704	21.69	206.08	1.88
	5	682	*	339.03	2783.99	682	62.77	276.58	0.00
17	1	859		493.15	86400.27	842	84.39	158.62	−1.98
	2	866		283.79	86400.41	851	56.61	132.50	−1.73
	3	850		655.98	86400.02	842	38.84	200.61	−0.94
	4	853		102.91	86400.38	845	7.38	279.73	−0.94
	5	845		263.56	86400.52	842	44.73	402.22	−0.36

($seconds_h$), and the total CPU time in seconds ($seconds_{tot}$). Because the branch-and-cut is not always run to completion, the value of z_{bc} may be higher than that of z_{TS}. In addition, asterisks indicate proven optimal solutions, and $\%gap$ evaluates the quality of the Tabu Search solution as $\%gap = 100$ (z_{TS} - z_{bc})/z_{bc}. The solutions found by the Tabu Search algorithm are very close to optimality, and are provably optimal in 33 out of 57 known optima. For half of the instances with 29 customers or more, the Tabu Search solution is better than that found by the branch-and-cut in a much higher CPU time (negative $\%gap$ values). On the other hand, for about one third of the instances with 25 customers or less the Tabu Search CPU time is higher than that of branch-and-cut and, in six of these cases, no optimal solution is found. The worse percentage error is 7.69, the best percentage improvement is 21.23. On average the Tabu Search solutions improve those obtained through branch-and-cut by 0.21%. The CPU times required by the Tabu Search algorithm are very reasonable. On average it takes 116 seconds in total, and 43 seconds to find the best solution, while for the branch-and-cut algorithm these values increase to 29 858 and 12 875 seconds, respectively.

In Tables 7.5 and 7.6 we summarize the results for the complete set of instances, both for sequential and unrestricted loading. The limit given to the Tabu Search heuristic is again the first occurrence between $2n^2v$ iterations and one hour of CPU time. The edge costs are computed as the rational Euclidean distances between any pair of vertices, without rounding them to integer values. One-customer routes are allowed, and fewer than v vehicles may be used. The entries z_s and z_u represent the values of the best solutions found for the sequential and unrestricted version, respectively. For each version, we also give the CPU time in seconds in which the best solution was found (sec_h) and the total CPU time in seconds taken by the algorithm (sec_{tot}).

The results are finally summarized in Table 7.5, in order to express some conclusions. The entries in the second column give the solution value obtained for Class 1, i.e., for pure CVRP instances. In the remaining columns we give, respectively for the sequential and unrestricted case, the average solution values for Classes 2-5 ($\overline{z}_{2-5}$), the average elapsed CPU time when the final solution was found ($seconds_h$) and the average total CPU time ($seconds_{tot}$). Both times are computed over the five considered classes.

The Tabu Search algorithm could find a feasible solution for all 360 instances. By comparing columns z_1 and $\overline{z}_{2-5}$ (unrestricted) one can observe that the inclusion of the loading constraint considerably worsens the solution values of CVRP (on average by 53.48%). Column $\overline{z}_{2-5}$ (sequential) shows that a relatively smaller further increase (on average by 4.15%) is produced by the constraint on sequential loading. The total CPU times are very close for the two versions, partially also due to the fact that for the largest instances with $n > 75$ the time limit of one hour was almost systematically reached. The time needed to find the best solution is lower for the unrestricted version (average $time_h = 758$ seconds) than for the sequential one (average $time_h = 936$ seconds).

7.5.1 Robustness

The robustness of the algorithm was tested by running it with different values of the parameters. These tests were performed on the instances of tables 7.3 and 7.4.

The algorithm is quite sensitive to the number of iterations allowed. As previously mentioned, the maximum number of iterations was experimentally determined as $2n^2v$, halting the execution, in any case, after one hour of CPU time. Other rules were tested, leading to a worse average ratio of solution values over computing times. In particular, replacing $2n^2v$ with a fixed limit of value 200 000 improved the average gap from -0.21% to -0.63%, and the number of negative gaps from 19 to 22, but the average computing time increased from 116 to 1547 seconds. Trying 100 000 iterations and 30 minutes halved the CPU time but decreased by one half the average improvement . Attempts with 300 000 iterations did not lead to any significant variation.

The algorithm proved to be particularly robust with respect to the policy adopted for the penalization factor γ, which was finally set to $\sqrt{nv}$, as in [143] and [32]. Other functions attempted, such as $\sqrt{nM}$, nv and nM, produced marginal variations. The approach proved to be also quite robust with respect to the length of the Tabu list, ϑ. The value $\vartheta = \log(nv)$ was finally chosen, but other functions of n and v produced similar results. Functions of the total number of items M proved instead to be a

Table 7.5: Performance of the Tabu Search on sequential and unrestricted instances (1).

Inst.		*sequential*			*unrestricted*			*Inst.*		*sequential*			*unrestricted*		
no	*cl*	z_s	sec_h	sec_{tot}	z_u	sec_h	sec_{tot}	*no*	*cl*	z_s	sec_h	sec_{tot}	z_u	sec_h	sec_{tot}
1	1	278.73	1.98	2.20	278.73	2.13	2.48	10	1	538.79	6.08	81.66	538.79	6.64	87.34
	2	301.45	4.94	6.22	284.42	1.25	4.92		2	715.51	49.26	72.50	701.25	43.19	92.36
	3	313.91	2.86	12.17	306.31	10.00	12.72		3	660.27	18.72	140.39	640.19	22.91	166.69
	4	296.75	1.06	8.38	296.94	1.98	9.89		4	769.73	59.00	195.06	761.66	49.17	201.63
	5	284.23	2.01	17.17	278.73	5.56	18.50		5	714.08	84.37	408.44	698.47	58.33	411.80
2	1	334.96	0.01	1.36	334.96	0.02	1.50	11	1	505.01	2.50	98.89	505.01	2.95	115.31
	2	347.73	1.56	1.89	334.96	0.05	2.03		2	754.62	23.63	76.61	722.58	20.19	95.78
	3	356.24	0.02	3.33	352.16	0.13	3.03		3	789.95	10.45	122.75	715.81	5.08	117.81
	4	342.00	0.31	4.83	342.00	0.00	5.70		4	899.46	98.42	232.50	859.18	175.94	216.53
	5	334.96	0.20	6.11	334.96	0.12	6.20		5	649.75	360.08	466.17	658.60	74.22	492.52
3	1	359.77	3.47	8.41	359.77	3.72	9.56	12	1	610.57	28.48	32.50	610.57	32.58	37.30
	2	411.24	7.50	9.86	387.70	0.58	11.75		2	641.84	24.41	45.20	627.43	12.01	46.14
	3	394.72	0.49	19.72	394.72	0.78	23.64		3	610.57	24.83	52.92	610.23	20.47	52.77
	4	376.83	5.27	25.69	368.56	0.83	23.12		4	664.76	39.80	124.69	630.59	80.03	103.86
	5	358.40	2.36	30.86	358.40	1.98	29.77		5	610.23	176.28	242.28	614.23	100.06	173.22
4	1	430.88	0.13	5.74	430.88	0.13	5.95	13	1	2006.34	29.88	161.61	2006.34	29.83	162.47
	2	440.94	0.67	7.74	430.88	0.81	9.41		2	2836.79	50.80	73.75	2616.19	9.11	155.05
	3	446.61	4.66	14.16	440.68	0.41	14.89		3	2625.82	30.27	186.41	2527.44	35.45	219.61
	4	455.25	1.03	19.30	447.37	0.30	15.73		4	2743.17	8.83	299.61	2725.50	64.50	281.48
	5	430.88	0.33	37.84	430.88	1.11	27.73		5	2542.34	125.05	842.66	2523.68	148.41	841.66
5	1	375.28	1.39	12.78	375.28	1.41	13.05	14	1	837.67	22.23	152.11	837.67	23.00	156.83
	2	390.62	7.42	17.78	379.94	9.58	19.61		2	1195.39	14.09	138.53	1079.51	34.86	205.94
	3	383.87	3.03	21.55	381.69	10.41	19.08		3	1156.33	154.88	235.31	1110.59	240.33	319.78
	4	386.47	4.78	28.91	383.87	0.28	24.59		4	1058.98	130.86	465.78	1014.75	331.52	462.47
	5	375.28	3.98	56.94	375.28	3.55	49.55		5	995.25	408.08	1205.56	986.02	1249.56	1683.14
6	1	495.85	0.33	8.69	495.85	0.34	9.63	15	1	837.67	1.75	182.50	837.67	2.17	204.56
	2	499.08	7.09	12.61	498.16	5.77	12.05		2	1143.73	4.67	119.78	1043.84	24.78	172.91
	3	504.68	12.95	15.30	504.68	4.09	15.86		3	1209.60	132.61	175.31	1173.83	7.75	167.00
	4	511.52	5.11	30.81	505.38	25.44	30.27		4	1311.09	14.73	364.27	1305.89	65.69	350.95
	5	495.85	0.06	29.95	495.85	0.33	26.09		5	1299.14	673.31	725.39	1281.97	683.31	983.89
7	1	568.56	0.49	22.59	568.56	0.53	26.53	16	1	698.61	2.75	98.98	698.61	2.95	102.81
	2	751.15	2.41	19.99	725.46	0.86	20.02		2	698.61	15.31	92.83	698.61	39.86	91.44
	3	702.59	9.28	35.22	702.59	14.72	38.00		3	698.61	18.17	135.23	698.61	35.41	129.12
	4	732.54	29.30	53.59	703.85	6.95	49.17		4	723.58	100.30	200.77	712.30	20.14	180.66
	5	701.31	35.97	133.77	669.47	8.25	108.52		5	698.61	3.55	258.05	698.61	4.09	296.73
8	1	568.56	0.48	36.23	568.56	0.55	39.86	17	1	862.62	59.03	162.77	862.62	62.72	173.42
	2	730.87	1.48	19.25	674.55	2.95	23.84		2	874.34	25.69	135.38	873.85	33.23	139.53
	3	771.29	14.09	37.78	741.12	5.41	27.44		3	862.62	72.98	198.84	862.62	168.59	195.50
	4	723.55	45.78	61.36	714.77	17.16	60.47		4	866.42	150.95	283.62	866.40	1.88	263.14
	5	665.20	101.97	263.89	649.52	29.84	155.09		5	862.62	136.00	350.45	862.62	58.25	319.48
9	1	607.65	0.36	13.52	607.65	0.37	13.98	18	1	723.54	81.86	587.30	723.54	85.50	620.88
	2	625.13	1.78	17.58	607.65	0.42	16.28		2	1129.51	172.58	307.75	1087.09	2.73	651.52
	3	638.31	23.47	36.88	638.31	7.84	37.09		3	1130.19	246.12	830.98	1111.11	16.55	858.48
	4	625.13	5.02	46.81	625.13	4.81	48.56		4	1209.25	197.11	948.66	1184.74	47.31	1342.09
	5	607.65	22.56	85.09	607.65	4.63	90.23		5	995.74	2134.86	3164.08	960.43	2794.42	3116.94

Table 7.6: Performance of the Tabu Search on sequential and unrestricted instances (2).

Inst.		*sequential*			*unrestricted*			*Inst.*		*sequential*			*unrestricted*		
no	*cl*	z_s	$time_h$	*time*	z_u	$time_h$	*time*	*no*	*cl*	z_s	$time_h$	*time*	z_u	$time_h$	*time*
19	1	524.61	253.61	1005.22	524.61	267.39	1057.78	28	1	1078.27	3079.98	3600.04	1078.27	3209.44	3600.43
	2	826.00	64.53	547.23	786.11	117.80	527.86		2	2872.77	3375.31	3600.01	2668.85	674.49	3600.02
	3	825.01	321.67	942.92	797.54	727.23	888.11		3	2929.11	2276.04	3600.03	2831.04	2069.15	3600.01
	4	866.91	757.77	1511.50	831.80	166.31	1548.77		4	2931.98	1808.95	3600.10	2820.11	2262.87	3600.09
	5	692.00	428.45	3600.50	673.53	1889.58	3600.00		5	2700.27	2344.37	3600.37	2633.25	2039.85	3600.2
20	1	241.97	325.00	2978.66	241.97	332.95	3039.27	29	1	1179.01	1834.45	3600.76	1179.01	1826.02	3600.15
	2	619.26	155.63	3072.60	570.59	339.13	2347.03		2	2658.72	30.25	3600.03	2465.92	358.27	3600.02
	3	581.94	674.53	3600.02	573.36	2532.78	3600.00		3	2390.00	227.12	3600.04	2359.09	1769.82	3600.04
	4	614.05	627.05	3600.00	609.89	28.94	3600.02		4	2732.84	208.64	3600.06	2582.33	153.93	3600.04
	5	512.00	2262.11	3600.14	504.84	1538.78	3600.16		5	2519.56	3512.22	3600.04	2489.99	2924.43	3600.02
21	1	688.18	2070.73	3600.11	688.18	2131.30	3600.02	30	1	1061.55	288.79	3600.95	1061.55	288.98	3600.59
	2	1183.37	2292.09	3405.70	1076.97	15.06	3600.03		2	2087.30	3475.11	3600.01	1999.52	807.34	3600.04
	3	1197.72	1512.28	3600.03	1196.67	13.06	3600.03		3	2194.36	1648.17	3600.01	2011.87	1015.65	3600.02
	4	1078.30	2486.66	3600.03	1045.60	53.35	3600.03		4	2164.01	2955.72	3600.01	2099.18	682.01	3600.07
	5	981.38	149.07	3600.05	945.60	87.69	3600.11		5	1859.14	1739.12	3600.03	1684.32	3133.06	3600.82
22	1	740.66	2210.10	3600.00	740.66	2236.52	3600.00	31	1	1464.04	1780.81	3600.12	1464.04	1783.46	3600.18
	2	1141.28	440.50	2909.05	1098.61	222.99	3322.56		2	2574.21	2247.11	3600.08	2464.92	2515.53	3600.43
	3	1162.24	322.99	3600.00	1095.02	41.53	3600.06		3	2592.07	3190.03	3600.09	2509.75	2790.35	3600.31
	4	1195.72	1480.80	3600.05	1134.32	350.87	3600.05		4	2808.08	2914.52	3600.00	2756.75	2495.12	3600.09
	5	1022.06	3414.88	3600.02	1021.87	3103.95	3600.05		5	2394.31	378.44	3602.25	2296.52	2294.4	3600.15
23	1	860.47	866.94	3600.03	860.47	849.13	3600.03	32	1	1352.61	2531.69	3601.09	1352.61	2501.72	3600.43
	2	1335.10	619.21	3600.03	1125.85	3053.64	3293.06		2	2652.73	1716.58	3600.11	2509.03	1070.12	3600.04
	3	1189.67	629.68	3600.06	1151.58	2378.39	3600.02		3	2576.98	1929.27	3600.00	2545.84	1042.58	3600.37
	4	1170.37	332.40	3600.01	1146.52	2779.11	3600.00		4	2816.35	2596.44	3600.03	2522.25	3306.78	3600.11
	5	1050.31	930.73	3600.01	994.94	1101.80	3600.02		5	2374.33	2752.16	3601.52	2368.60	402.65	3601.25
24	1	1048.91	2371.00	3598.78	1048.91	2322.05	3524.00	33	1	1361.51	788.56	3600.59	1361.51	798.92	3600.73
	2	1382.85	1487.15	2224.01	1285.18	1717.61	1956.28		2	2605.16	1948.88	3600.03	2472.05	649.72	3600.01
	3	1202.90	3370.61	3600.01	1163.25	3185.72	3600.00		3	2687.43	2429.47	3600.02	2595.28	2008.61	3600.04
	4	1322.48	3449.18	3600.03	1215.46	39.19	3600.02		4	2851.39	3253.74	3600.28	2664.51	3240.97	3600.09
	5	1085.48	2534.44	3600.14	1086.60	6.19	3600.05		5	2298.21	2685.33	3602.13	2284.15	2517.94	3600.22
25	1	830.26	3597.63	3600.29	832.97	1981.18	3600.02	34	1	858.94	1941.88	3604.38	858.94	1920.81	3606.48
	2	1567.22	2298.96	3600.01	1501.06	128.13	3600.00		2	1573.77	2103.24	3600.03	1496.45	346.54	3600.01
	3	1495.01	2411.59	3600.04	1445.56	1433.43	3600.00		3	1641.66	861.03	3600.02	1510.97	652.03	3600.03
	4	1542.21	1139.68	3600.04	1543.46	270.61	3600.00		4	1586.18	3586.77	3600.46	1500.51	1416.41	3600.09
	5	1318.08	2234.52	3600.09	1254.27	2215.43	3600.14		5	1382.64	2429.16	3600.16	1356.30	2459.66	3600.11
26	1	819.56	355.87	3600.07	819.56	353.75	3600.11	35	1	992.86	766.72	3600.32	992.86	798.39	3600.27
	2	1427.04	1296.60	3600.02	1360.54	925.89	3600.03		2	1846.79	2028.03	3600.05	1684.13	3493.17	3600.05
	3	1447.94	2781.46	3600.02	1444.67	925.08	3600.01		3	1852.52	2263.58	3600.03	1761.12	203.98	3600.08
	4	1594.35	2099.27	3600.06	1478.32	2710.33	3600.04		4	2743.16	3271.25	3600.14	2138.60	2884.98	3600.03
	5	1417.61	1239.90	3601.52	1334.41	954.39	3600.08		5	1499.30	2785.94	3600.46	1477.35	2928.2	3601.67
27	1	1099.95	985.18	3600.02	1099.95	986.09	3600.04	36	1	678.87	1530.91	3603.94	677.16	3481.12	3600.19
	2	1610.60	3017.93	3600.02	1443.10	19.36	3600.01		2	1994.16	2217.94	3600.00	1888.66	973.67	3600.03
	3	1550.71	1883.15	3600.01	1508.84	24.53	3600.01		3	2082.29	2987.02	3600.00	2118.57	111.22	3600.03
	4	1540.83	66.88	3600.07	1460.10	1373.72	3600.08		4	1954.92	2840.08	3600.13	1889.55	3455.42	3600.06
	5	1394.74	588.06	3600.06	1388.66	202.76	3600.05		5	1755.27	3555.43	3600.33	1742.73	3307.56	3600.16

Table 7.7: Performance of the Tabu Search heuristic. Real edge costs.

		Unrestricted			*Sequential*		
Inst.	z_1 (CVRP)	$\overline{z}_{2-5}$	$seconds_h$	$seconds_{tot}$	$\overline{z}_{2-5}$	$seconds_h$	$seconds_{tot}$
1	278.73	291.60	4.18	9.70	299.09	2.57	9.23
2	334.96	341.02	0.06	3.69	345.23	0.42	3.50
3	359.77	377.35	1.58	19.57	385.30	3.82	18.91
4	430.88	437.45	0.55	14.74	443.42	1.36	16.96
5	375.28	380.20	5.05	25.18	384.06	4.12	27.60
6	495.85	501.02	7.19	18.78	502.78	5.11	19.47
7	568.56	700.34	6.26	48.45	721.90	15.49	53.03
8	568.56	694.99	11.18	61.34	722.73	32.76	83.70
9	607.65	619.69	3.61	41.23	624.06	10.64	39.98
10	538.79	700.39	36.05	191.97	714.90	43.49	179.61
11	505.01	739.04	55.68	207.59	773.45	99.02	199.38
12	610.57	620.62	49.03	82.66	631.85	58.76	99.52
13	2006.34	2598.20	57.46	332.05	2687.03	48.97	312.81
14	837.67	1047.72	375.85	565.63	1101.49	146.03	439.46
15	837.67	1201.38	156.74	375.86	1240.89	165.41	313.45
16	698.61	702.03	20.49	160.15	704.85	28.02	157.17
17	862.62	866.37	64.93	218.21	866.50	88.93	226.21
18	723.54	1085.84	589.30	1317.98	1116.17	566.51	1167.75
19	524.61	772.25	633.66	1524.50	802.48	365.21	1521.47
20	241.97	564.67	954.52	3237.30	581.81	808.86	3370.28
21	688.18	1066.21	460.09	3600.04	1110.19	1702.17	3561.18
22	740.66	1087.46	1191.17	3544.54	1130.33	1573.85	3461.82
23	860.47	1104.72	2032.41	3538.63	1186.36	675.79	3600.03
24	1048.91	1187.62	1454.15	3256.07	1248.43	2642.48	3324.59
25	830.26	1436.09	1205.76	3600.03	1480.63	2336.48	3600.09
26	819.56	1404.49	1173.89	3600.05	1471.74	1554.62	3600.34
27	1099.95	1450.18	521.29	3600.04	1524.22	1308.24	3600.04
28	1078.27	2738.31	2051.16	3600.15	2858.53	2576.93	3600.11
29	1179.01	2474.33	1406.49	3600.05	2575.28	1162.54	3600.19
30	1061.55	1948.72	1185.41	3600.31	2076.20	2021.38	3600.20
31	1464.04	2506.99	2375.77	3600.23	2592.17	2102.18	3600.51
32	1352.61	2486.43	1664.77	3600.44	2605.10	2305.23	3600.55
33	1361.51	2504.00	1843.23	3600.22	2610.55	2221.20	3600.61
34	858.94	1466.06	1359.09	3601.34	1546.06	2184.42	3601.01
35	992.86	1765.30	2061.74	3600.42	1985.44	2223.10	3600.20
36	678.87	1909.88	2265.80	3600.10	1946.66	2626.28	3600.88

bad choice. As for the size p of the neighborhood, we tested the values 5, 9, 10, 11 and 20. The best choice proved to be $p = 10$. The results were close for $p = 9$ and $p = 11$, but definitely worse for $p = 5$ and $p = 20$.

Different rules were tried for the updating of parameters α and β, but none managed to find the results obtained through the rule described in Section 7.3.3. The behavior of the algorithm also consistently changed when changing the maximum limit given to λ, the maximum number of calls to TP_{2SL} and resp. TP_{2UL}. The values 1, 2, 3, 4, 5 and 10 were attempted, and the value 4 was chosen. With the value 5, the CPU time increased by 3%, improving the average gap only by 0.02%. An almost symmetric effect was produced by using the value 3.

The periodic intensification (see Section 7.4.1) is performed every ω iterations. Several values were tested for ω: 10, 5, 4, 3, 2 and 1. It turned out that frequent executions are useful for small-size instances. The best results were obtained by setting ω to 1 for $n \leq 40$, and to 5 for larger values.

7.6 Conclusions

We have considered an extension of the classical vehicle routing problem in which two-dimensional packing constraints are introduced. This problem features of two classical combinatorial optimization problems. We have developed, implemented and tested a Tabu Search algorithm which was applied to a large test set obtained by modifying instances from the literature. On instances solved optimally by branch-and-cut, the proposed algorithm also finds an optimal solution in 33 cases out 57. On instances for which the branch-and cut was interrupted before completion, our heuristic finds feasible solutions that are on the average 0.21% better. The Tabu Search algorithm was also applied to a larger set of 360 instances for which the optimum was not known. It succeeded in identifying a feasible solution in all cases. Our results also show that the introduction of a loading constraint considerably increases solution costs.

Bibliography

[1] E. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120. J. Wiley & Sons, Chichester, 1997.

[2] E. Aarts and J. K. Lenstra (eds.). *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.

[3] A. Amaral and A.N. Letchford. An improved upper bound for the two-dimensional non-guillotine cutting problem. Technical report, Lancaster University, UK, available at http://www.lancs.ac.uk/staff/letch/ngc.doc, 2001.

[4] M. Arenales and R. Morabito. An and/or-graph approach to the solution of two-dimensional guillotine cutting problems. *European Journal of Operational Research*, 84:599–617, 1995.

[5] L. Babel, H. Kellerer, and V. Kotov. The k-partitioning problem. *Mathematical Methods of Operations Research*, 47:59–82, 1998.

[6] B. S. Baker, D. J. Brown, and H. P. Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2:348–368, 1981.

[7] B. S. Baker, E. G. Coffman, Jr., and R. L. Rivest. Orthogonal packing in two dimensions. *SIAM Journal on Computing*, 9:846–855, 1980.

[8] D. S. Baker and J. S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12:508–525, 1983.

[9] J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette, editor, *Proceedings of the 2nd international conference on genetic algorithms*, pages 14–21. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.

[10] G. Barbarasoglu and D. Ozgur. A tabu search algorithm for the vehicle routing problem. *Computers and Operations Research*, 26:255–270, 1999.

[11] J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

[12] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33:49–64, 1985.

[13] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[14] J.E. Beasley. A population heuristic for constrained two-dimensional non-guillotine cutting. *European Journal of Operational Research*, 2003 (to appear).

[15] B. E. Bengtsson. Packing rectangular pieces – a heuristic approach. *The Computer Journal*, 25:353–357, 1982.

[16] J. O. Berkey and P. Y. Wang. Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[17] M.A. Boschetti, E. Hadjiconstantinou, and A. Mingozzi. New upper bounds for the two-dimensional othogonal non guillotine cutting stock problem. *IMA Journal of Management Mathematics*, 13:95–119, 2002.

[18] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. *4OR*, 1:27–42, 2003.

[19] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. *4OR*, 1:135–147, 2003.

[20] D. J. Brown. An improved BL lower bound. *Information Processing Letters*, 11:37–39, 1980.

[21] V. Campos, M. Laguna, and R. Martí. Scatter search for the linear ordering problem. In Glover F. Corne D., Dorigo M., editor, *New Ideas in Optimization*. Wiley, 1999.

[22] A. Caprara and M. Monaci. On the 2-dimensional knapsack problem. *Operations Research Letters*, 2003 (to appear).

[23] L. Cavique, C. Rego, and I. Themido. A scatter search algorithm for the maximum clique problem. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 227–244. Kluwer Academic Publishers, Boston, 2001.

[24] B. Chazelle. The bottom-left bin packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, 32:697–707, 1983.

[25] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[26] G. Clarke and J.V. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[27] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:801–826, 1980.

[28] E. G. Coffman, Jr., M.R. Garey, and D.S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978.

[29] E. G. Coffman, Jr., G.S. Lueker, and A.H.G. Rinnooy Kan. Asymptotic methods in the probabilistic analysis of sequencing and packing heuristics. *Management Science*, 34:266–290, 1988.

[30] J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. New heuristics for the vehicle routing problem. In A. Langevin and D. Riopel, editors, *Logistics Systems: Design and Optimization*. Kluwer, Boston, 2005 (to appear).

[31] J.-F. Cordeau and G. Laporte. Tabu search heuristics for the vehicle routing problem. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, pages 145–163. Kluwer, Boston, 2004.

[32] J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.

[33] J.F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30:105–119, 1997.

[34] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6:80, 1959.

[35] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 162–64. Morgan Kaufmann, 1985.

[36] L. Davis, editor. *A Handbbok of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[37] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[38] M. Dell' Amico and S. Martello. Bounds for the cardinality constrained $P||C_{\max}$ problem. *Journal of Scheduling*, 4:123–138, 2001.

[39] M. Dell'Amico, M. Iori, and S. Martello. Heuristic algorithms and scatter search for the cardinality constrained $P||C_{\max}$ problem. *Journal of Heuristics*, 10:169–204, 2004.

[40] M. Dell'Amico, M. Iori, M. Monaci, and S. Martello. Lower bounds and heuristic algorithms for the k_i partitioning problem. *European Journal of Operational Research, Special Issue on Stochastic and Heuristic Methods in Optimization*, 2004 (to appear).

[41] M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200, 1995.

[42] J.J. Dongarra. Performance of various computers using standard linear equations software. Working paper, Computer Science Department, University of Tennesse, 2003.

[43] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In Glover F. Corne D., Dorigo M., editor, *New Ideas in Optimization*, pages 11–32. Mc Graw Hill, London, UK, 1999.

[44] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for dicrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[45] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 251–286. Kluwer Academic Publishers, Boston, 2003.

[46] K. A. Dowsland and W. B. Dowsland. Packing problems. *European Journal of Operational Research*, 56(1):2–14, 1992.

[47] K.A. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399, 1993.

[48] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[49] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and Packing (C&P). In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 393–413. John Wiley & Sons, Chichester, 1997.

[50] S.M. Fatemi-Ghomi and F. Jolai-Ghazvini. A pairwise interchange algorithm for parallel machine scheduling. *Production Planning and Control*, 9:685–689, 1998.

[51] S.P. Fekete and J. Schepers. On more-dimensional packing II: Bounds. Technical Report ZPR97-289, Mathematisches Institut, Universität zu Köln, 1997.

[52] S.P. Fekete and J. Schepers. On more-dimensional packing I: Modeling. *Mathematics of Operations Research*, 2003 (to appear).

[53] S.P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithms. *Operations Research*, 2003 (to appear).

[54] P. Festa and M.G.C. Resende. Grasp: an annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, Boston, 2001.

[55] G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, 19:312–320, 1998.

[56] A. Frangioni, E. Necciari, and M. G. Scutellá. A multi-exchenge neighborhood for minimum makespan machine scheduling problems. *Technical report*, 2000.

[57] M. Gendreau. An introduction to tabu search. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 37–54. Kluwer Academic Publishers, Boston, 2003.

[58] M. Gendreau, A. Hertz, and Laporte G. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[59] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[60] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search approach to vehicle routing problems with two-dimensional loading constraints. Technical Report OR/04/1, DEIS, Università di Bologna, 2004.

[61] P. C. Gilmore and R. E. Gomory. Multistage cutting problems of two and more dimensions. *Operations Research*, 13:94–119, 1965.

[62] F. Glover. Parametric combinations of local job shop rules. In *ONR Research Memorandum no. 117*. GSIA, Carnegie Mellon University, Pittsburgh, Pa, 1963.

[63] F. Glover. A multiphase dual algorithm for the zero-one integer programming problem. *Operation Research*, 13(6):879, 1965.

[64] F. Glover. Heurisic for integer programming using surrogate constraints. *DS*, 8:156–166, 1977.

[65] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[66] F. Glover. Tabu seach – part i. *ORSA Journal on Computing*, 1:190–206, 1989.

[67] F. Glover. Tabu seach and adaptive memory programming – advances, applications and challenges. In R.S. Barr, R.V. Helagson, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, Boston, 1996.

[68] F. Glover. A template for scatter search and path relinking. In J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Lecture Notes in Computer Science*, volume 1363, pages 1–45. Springer, 1997.

[69] F. Glover. Scatter search and path relinking. In Glover F. Corne D., Dorigo M., editor, *New Ideas in Optimization*. Wiley, 1999.

[70] F. Glover and G.A. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston, 2003.

[71] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[72] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 1–36. Kluwer Academic Publishers, Boston, 2003.

[73] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Foundations and advanced designs. In G. C. Onwubolu and B. V. Babu, editors, *New Optimization Techniques in Engineering*. Springer-Verlag, Heidelberg, 2004 (in print).

[74] I. Golan. Performance bounds for orthogonal oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 10:571–582, 1981.

[75] D. E. Goldberg and R. L. Lingle. Alleles, loci and the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, pages 154–159. Erlbaum, 1985.

[76] A. Gomez and D. De La Fuente. Resolution of strip-packing problems with genetic algorithms. *Journal of the Operational Research Society*, 51:1289–1295, 2000.

[77] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[78] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[79] J.J. Grefenstette. Incorporating problem-specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60. Morgan Kauffmann Publishers, 1987.

[80] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, 83:39–56, 1995.

[81] E. Hadjiconstantinou and M. Iori. A genetic algorithm for the two-dimensional knapsack problem. Technical Report OR/02/8, DEIS, Università di Bologna, 2002.

[82] R.W. Haessler and P.E. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150, 1991.

[83] J.P. Hamiez and J.K. Hao. Scatter search for graph coloring. LNCS *Series*, 2001 (to appear).

[84] P. Hansen and N. Mladenović. Variable neighborhood search. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 145–184. Kluwer Academic Publishers, Boston, 2003.

[85] P. Healy, M. Creavin, and A. Kuusik. An optimal algotihm for rectangle placement. *Operations Research Letters*, 24:73–80, 1999.

[86] D. Henderson, S.H. Jacobson, and A.W. Johnson. The theory and practice of simulated annealing. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 287–320. Kluwer Academic Publishers, Boston, 2003.

[87] J.C. Hertz. A recursive computational procedure for two - dimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

[88] M. Hifi. The strip cutting/packing problem: incremental substrip algorithms-based heuristics. *Pesquisa Operacional, Special Issue on Cutting and Packing Problems*, 19(2):169–188, 1999.

[89] M.S. Hillier and M.L. Brandeau. Optimal component assignment and board grouping in printed circuit board manufacturing. *Operations Research*, 46(5):675–689, 1998.

[90] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of ACM*, 34(1):144–162, 1987.

[91] J. Holland. *Adaptation in Natural and Artificial Systems*. Michigan Press, 1975.

[92] A. Hoogeveen, J.K. Lenstra, and S.L. van de Velde. Sequencing and scheduling. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 181–197. Wiley, Chichester, 1997.

[93] J.J. Hopfield. Neural netwroks and physical systems with emergent collective computational capabilities. In *Proceedings of the National Academy of Sciences*, volume 79, pages 2554–2558, 1982.

[94] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128(1):34–57, 2001.

[95] S. Høyland. Bin-packing in 1.5 dimension. In R.G. Karlsson and A. Lingas, editors, *Lecture Notes in Computer Science*, pages 129–137. Springer-Verlag, Berlin, 1988.

[96] R. Hübsher and F. Glover. Applying tabu search with influential diversification to multiprocessor scheduling. *COR*, 21:877–889, 1994.

[97] M. Iori, S. Martello, and M. Monaci. Metaheuristic algorithms for the strip packing problem. In P. Pardalos and V. Korotkich, editors, *Optimization and Industry: New Frontiers*, pages 159–179. Kluwer Academic Publisher, 2003.

[98] M. Iori, J.J. Salazar Gonzalez, and D. Vigo. An exact approach for the symmetric capacitated vehicle routing problem with two dimensional loading constraints. Technical Report OR/03/04, DEIS, Università di Bologna, 2003.

[99] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88:165–181, 1996.

[100] J.J. Jimeno, I. Gutiérrez, and E. Mokotoff. List scheduling algorithms to minimize the makespan on identical parallel machines. *TOP*, 9(2):243–269, 2001.

[101] M. Laguna. Scatter search. In P. M. Pardalos and M. G. C. Resende, editors, *HandBOOK of Applied Optimization*, pages 183–193. Oxford University Press, 2002.

[102] K.K. Lai and J.W.M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Computers and Industrial Engineering*, 32:115–127, 1997.

[103] K.K. Lai and J.W.M. Chan. An evolutionary algorithm for the rectangular cutting stock problem. *International Journal of Industrial Engineering*, 4:130–139, 1997.

[104] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In Graves S.C. et al., editor, *HandBOOKs in Operations Research and Managemeny Science*, pages 445–522. North-Holland, Amsterdam, 1993.

[105] T.W. Leung, C.H. Yung, and M.D. Troutt. Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. *Computers and Industrial Engineering*, 40:201–214, 2001.

[106] D. Liu and H. Teng. An improved bl-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research*, 112:413–420, 1999.

[107] A. Lodi. Multi-dimensional packing by tabu search. *Studia Informatica Universalis*, 2:111–126, 2002.

[108] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141(2):241–252, 2001.

[109] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141:241–252, 2002.

[110] A. Lodi, S. Martello, and D. Vigo. Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 125–139. Kluwer Academic Publishers, Boston, 1998.

[111] A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112:158–166, 1999.

[112] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11:345–357, 1999.

[113] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 2001 (to appear).

[114] A. Lodi, S. Martello, and D. Vigo. Tspack: A unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, 2002 (to appear).

[115] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[116] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48:256–267, 2000.

[117] S. Martello and P. Toth. Worst-case analysis of greedy algorithms for the subset-sum problem. *Mathematical Programming*, 28:198–205, 1984.

[118] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990.

[119] S. Martello and P. Toth. An exact algorithm for the two-constraint 0-1 knapsack problem. *Operations Research*, 51:826–835, 2003.

[120] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.

[121] Martí. Multi-start methods. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 355–368. Kluwer Academic Publishers, Boston, 2003.

[122] W.S. Mc Culloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[123] A.S. Mendes, F.M. Muller, P.M. Franca, and P. Moscato. Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control*, 13:143–154, 2002.

[124] N. Mladenović, M. Labbé, and P. Hansen. Solving the $p-$center problem by tabu search and variable neighborhood search. *Networks*, 2001 (to appear).

[125] E. Mokotoff. Parallel machine scheduling problems: a survey. *Asia-Pacific Journal of Operational Research*, 18:193–242, 2001.

[126] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Publishers, Boston, 2003.

[127] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[128] D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. Technical Report 03/1, Department of Computer Science, University of Copenhagen, 2003.

[129] P. W. Poon and J. N. Carter. Genetic algorithm crossover operators for ordering applications. *Computers and Operations Research*, 22(1):135–147, 1995.

[130] J.Y. Potvin and K.A. Smith. Artificial neural networks for combinatorial optimization. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 429–456. Kluwer Academic Publishers, Boston, 2003.

[131] C. Reeves. Hybrid genetic algorithms for the bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[132] C. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3), 1997.

[133] C. Reeves. Genetic algorithms. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 55–82. Kluwer Academic Publishers, Boston, 2003.

[134] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.

[135] C. Rego and C. Roucairol. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory and Applications*, pages 661–675. Kluwer, Boston, 1996.

[136] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–250. Kluwer Academic Publishers, Boston, 2003.

[137] Y. Rochat and E.D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

[138] G. Scheithauer and J. Terno. Modeling of packing problems. *Optimization*, 28:63–84, 1993.

[139] D. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10:37–40, 1980.

[140] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26:401–409, 1997.

[141] J.Y. Suh and D. Van Gucht. Incorporating heuristic information into genetic search. In *Proceedings of the Second International conference on Genetic Algorithms and their Application*, pages 100–107, Cambridge, Massachussetts, 1982.

[142] P.E. Sweeney and E.R. Paternoster. Cutting and packing problems: A categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43:691–706, 1992.

[143] E.D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[144] P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

[145] P. Toth and D. Vigo. The granular tabu search (and its application to the vehicle routing problem). *INFORMS Journal on Computing*, 15(4):333–346, 2003.

[146] C. Voudouris and E.P.K. Tsang. Guided local search. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 185–218. Kluwer Academic Publishers, Boston, 2003.

[147] P.Y. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31:573–586, 1983.

[148] J. Xu and J.P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30:379–393, 1996.