This is a pre print version of the following article:

Lower bounds and heuristic algorithms for the k_i-partitioning problem / Dell'Amico, M., Iori, M., S., M., M., M.. - In: EUROPEAN JOURNAL OF OPERATIONAL RESEARCH. - ISSN 0377-2217. - STAMPA. - 171:3(2006), pp. 725-742. [10.1016/j.ejor.2004.09.002]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

16/10/2026 22:35

(Article begins on next page)

Lower Bounds and Heuristic Algorithms for the k_i-Partitioning Problem

Mauro Dell'Amico

DISMI, Università di Modena e Reggio Emilia, Italy (dellamico@unimore.it)

Manuel Iori, Silvano Martello, Michele Monaci

DEIS, Università di Bologna, Italy (miori,smartello,mmonaci@deis.unibo.it)

Abstract

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.

1 Introduction

Given n *items* I_j $(j = 1, \ldots, n)$, each characterized by an integer positive *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 [1] 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 positive *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 [1], Dell'Amico and Martello [6] and Dell'Amico, Iori and Martello [4].

The *Bin Packing Problem* (*BPP*) too is related to k_i-*PP*. Here we are given n items, each having an associated integer positive *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 [2], Hochbaum and Shmoys [16], Dell'Amico and Martello [5]) and $P|\#\leq k|C_{\max}$ (Dell'Amico and Martello [6]).

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 2 we present lower bounds and reduction criteria. In Section 3 we examine generalizations of heuristic algorithms and of a scatter search approach. In Section 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 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.

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

$$\min z \tag{1}$$

$$\sum_{j=1}^{n} w_j x_{ij} \leq z \quad (i=1,\ldots,m) \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (j=1,\ldots,n) \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} \leq k_i \quad (i=1,\ldots,m) \tag{4}$$

$$x_{ij} \in \{0,1\} \quad (i=1,\ldots,m; j=1,\ldots,n) \tag{5}$$

where variable z represents the maximum weight of a subset.

In the following we will denote by

$$free = \sum_{i=1}^{m} k_i - n \tag{6}$$

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 (1)-(5),

$$W_i = \sum_{j=1}^{n} w_j x_{ij} \quad (i = 1, \ldots, m) \tag{7}$$

$$card_i = \sum_{j=1}^{n} x_{ij} \quad (i = 1, \ldots, m) \tag{8}$$

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 3.1, are performed at the beginning of the scatter search approach described in Section 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 4.

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

$$L_{|\#\leq k_m|} = \max(L_3^k, L_{BKK}, L_{HS}) \tag{9}$$

the best among three bounds from the literature, used in Dell'Amico, Iori and Martello [4] 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 [4] and in Dell'Amico and Martello [6], Babel, Kellerer and Kotov [1] and Hochbaum and Schmoys [16].

The following lower bounds explicitly take into account cardinality constraints (4).

Theorem 1. *Given any instance of* k_i-PP*, the value*

$$L_1 = w_1 + \sum_{j=n-k_1+free+2}^{n} w_j \tag{10}$$

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 (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 (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 $\beta < \alpha$.) $\triangle$

Theorem 2. *Given any instance of* k_i-PP*, the value*

$$L_2 = \max_{\ell=1,\ldots,m} \left\{ \left\lceil \sum_{j=n-k(\ell)+free+1}^{n} w_j/\ell \right\rceil \right\}, \tag{11}$$

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 a valid lower bound for any ℓ value, and the validity of (11) follows. $\triangle$

If the summation in (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

$$L_3 = \max\left(w_m + w_{m+1}, \left\lceil \sum_{j=1}^{n} w_j/m \right\rceil \right) \tag{12}$$

Our overall combinatorial bound is thus

$$L_C = \max(L_{|\#\leq k_m|}, L_1, L_2, L_3) \tag{13}$$

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 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 1: Reduction1

Reduction1 can be performed before any other computation. Once a lower bound L (e.g., L_C, see (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 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 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 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. [20], Hoogeveen, Lenstra and van de Velde [17] and Mokotoff [24]). 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 [15], that orders the jobs by non-increasing processing times. The probabilistic analysis in Coffman, Lueker and Rinnooy Kan [3] 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 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 [5], while methods using the second strategy are the *Multi Fit* algorithm (*MF*) proposed by Coffman, Garey and Johnson [2], and the ϵ-dual method by Hochbaum and Shmoys [16].

- *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 [25]).

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 $P||C_{\max}$ arising, 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 [9] introduced a polynomial improvement algorithm called *0/1 interchange*. Hübscher and Glover [18] proposed a tabu search algorithm, Fatemi-Ghomi and Jolai-Ghazvini [7] a local search approach based on 2-exchanges. Mendes et al. [23] compared a tabu search approach with a memetic algorithm, while Frangioni, Necciari and Scutellà [8] 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 [4].

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.

3.1 Constructive Heuristics

We obtained heuristics for k_i-*PP* by generalizing methods for $P||C_{max}$. The following algorithms were obtained (see (7) and (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 [25] 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 [5]. 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 positive 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 [21], 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 [21].

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.

3.2 Scatter Search

Scatter Search is a metaheuristic technique whose founding ideas go back to the seminal works of Glover [10, 11] in the early Sixties. Apparently these ideas were neglected for more than 20 years and resumed in Glover [12], 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 [13]) 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 [19], Glover, Laguna and Martí [14]) that was already used in Dell'Amico, Iori and Martello [4] for $P|\#\leq k|C_{\max}$, summarized in Figure 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: 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

$$\delta_j(s,t) = \begin{cases} 1 & \text{if } I_j \text{ is assigned to the same subset in solutions } s \text{ and } t; \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

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

$$d(s) = \min_{t \in R} \left\{ \sum_{j=1}^{\tau} \delta_j(s,t) \right\} \tag{15}$$

with $\tau = \min(2m, n)$, i.e., only the items with largest weight are relevant in this evaluation.

Intensification. Given a solution s, we apply a sequence of re-optimization procedures obtained by generalizing those introduced in Dell'Amico, Iori and Martello [4] 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 [14]), 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 [14]) 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 (14) and (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.

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, 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 4.2 to obtain a lower bound for k_i-*PP* through column generation.

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

$$\mathcal{G}^r = \{S_i : k_i \geq \kappa_r\} \tag{16}$$

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

$$\mathcal{P}^r(c) = \{P \subseteq \{I_1, \ldots, I_n\} : \kappa_{r-1} < |P| \leq \kappa_r \text{ and } \sum_{I_j \in P} w_j \leq c\} \tag{17}$$

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 (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

$$x_P = \begin{cases} 1 & \text{if pattern } P \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \qquad (P \in \mathcal{P}^r(c), \kappa_r \in K) \tag{18}$$

and

$$y_j = \begin{cases} 1 & \text{if item } I_j \text{ is not assigned to any subset} \\ 0 & \text{otherwise} \end{cases} \qquad (j = 1, \ldots, n) \tag{19}$$

We obtain the ILP model

$$SSPK(c) \qquad \min \sum_{j=1}^{n} y_j \tag{20}$$

$$\sum_{\kappa_r \in K} \sum_{P \in \mathcal{P}_j^r(c)} x_P + y_j = 1 \quad (j = 1, \ldots, n) \tag{21}$$

$$\sum_{i \geq r} \sum_{P \in \mathcal{P}^i(c)} x_P \leq |\mathcal{G}^r| \quad (\kappa_r \in K) \tag{22}$$

$$y_j \in \{0, 1\} \quad (j = 1, \ldots, n) \tag{23}$$

$$x_P \in \{0, 1\} \quad (P \in \mathcal{P}^r(c), \kappa_r \in K) \tag{24}$$

Constraints (21) impose that each item I_j is assigned to exactly one subset, or not assigned at all. Constraints (22) impose, for each $\kappa_r \in K$, the selection of at most $|\mathcal{G}^r|$ patterns (see (16)) among those containing κ_r items or more. Objective function (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.

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 (20)-(22) and

$$y_j \geq 0 \quad (j = 1, \ldots, n) \tag{25}$$

$$x_P \geq 0 \quad (P \in \mathcal{P}^r(c), \kappa_r \in K) \tag{26}$$

(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 (21) and (22), respectively, we obtain the dual:

$$\max \sum_{j=1}^{n} \pi_j - \sum_{\kappa_r \in K} |\mathcal{G}^r| \, \sigma_r \tag{27}$$

$$\sum_{j\in P}\pi_j - \sum_{i\le r}\sigma_i \le 0 \quad (P\in\mathcal{P}^r(c),\kappa_r\in K) \tag{28}$$

$$\pi_j \le 1 \quad (j=1,\dots,n) \tag{29}$$

$$\sigma_r \ge 0 \quad (\kappa_r\in K) \tag{30}$$

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 (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 (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 $\sum_{i\le r}\sigma_i^*$ for some $\kappa_r\in K$. Thus, for a given value $\kappa_r\in K$, our *slave* problem can be formulated as

$$\max \sum_{j=1}^{n}\pi_j^*\xi_j \tag{31}$$

$$\sum_{j=1}^{n} w_j\xi_j \le c \tag{32}$$

$$\sum_{j=1}^{n}\xi_j \le \kappa_r \tag{33}$$

$$\xi_j \in \{0,1\} \quad (j=1,\dots,n) \tag{34}$$

whose optimal solution identifies the pattern $P\in\mathcal{P}^r(c)$ with maximum profit, namely $P=\{I_j : \xi_j=1\}$. Problem (31)-(34) is a *0-1 Knapsack Problem* ((31), (32), (34)) with an additional cardinality constraint. Since the w_j values are positive, 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 [22] 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 (28) is violated.

5 Computational experiments

The overall algorithm introduced in the previous sections,

Step 1. lower bounds, reduction, constructive heuristics (Sections 2 and 3.1);

Step 2. scatter search (Section 3.2);

Step 3. column generation (Section 4),

was 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 [6], and are as follows (See Table 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 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 4.

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 1: Parameters used for the weight classes

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 2: Parameters used for the cardinality 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 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

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 4: Data generation for Classes $k7$–$k9$

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 and 4 present the overall performance of lower bounds and heuristic algorithms. Since it turned out that the results within each triple of classes ($w1$-$w3$, $w4$-$w6$, $w7$-$w9$, $k1$-$k3$, $k4$-$k6$, $k7$-$k9$) were very similar to each other, we only report in the tables the overall results for the "middle" class of each triple. In Table 3 the entries give, for the selected weight classes, the average performance over all cardinality classes, while in Table 4 they give, for the selected cardinality classes, the average performance over all weight classes. Both Tables 3 and 4 report the behavior of lower bounds $L_{|\#\leq k_m|}$, L_C (see Section 2) and L_{CG} (see Section 4), of constructive heuristics *LPT*-k_i, *GH*-k_i and *MS*-k_i (see Section 3.1), and of the scatter search algorithm of Section 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;
- $\%gap$ = 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 below, in Tables 6 and 7.)

Table 3: Overall performance of lower bounds and heuristics on selected 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	2605	2615	2790	100	225	1741	2760	2790
$w2$	*#opt*	2393	2402	2509	100	225	1709	2482	2509
	%gap	0.05	0.04	0.01	3.21	1.14	0.57	0.01	0.01
	#best	2618	2625	2790	132	253	1780	2766	2790
$w5$	*#opt*	2399	2406	2501	132	252	1742	2478	2501
	%gap	0.05	0.04	0.01	3.17	1.13	0.56	0.01	0.01
	#best	2605	2614	2790	111	224	1711	2767	2790
$w8$	*#opt*	2393	2400	2496	111	224	1674	2477	2496
	%gap	0.05	0.04	0.01	3.21	1.15	0.60	0.01	0.01
average	*#best*	2609.2	2617.2	2790.0	115.1	230.3	1745.3	2764.4	2790.0
$w1$-$w9$	*#opt*	2396.3	2403.5	2503.4	115.1	230.0	1707.0	2480.8	2503.4
	%gap	0.05	0.04	0.01	3.17	1.12	0.58	0.01	0.01

Table 4: Overall performance of lower bounds and heuristics on selected 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	2643	2643	2790	57	122	802	2762	2790
$k2$	*#opt*	2351	2351	2439	57	122	800	2414	2439
	%gap	0.04	0.04	0.01	1.70	1.31	0.96	0.01	0.01
	#best	2650	2650	2790	117	213	2154	2766	2790
$k5$	*#opt*	2415	2415	2487	117	213	2057	2470	2487
	%gap	0.04	0.04	0.01	1.19	0.74	0.19	0.01	0.01
	#best	2605	2628	2790	156	420	2227	2778	2790
$k8$	*#opt*	2412	2431	2519	156	420	2216	2510	2519
	%gap	0.05	0.03	0.01	5.23	0.64	0.34	0.01	0.01
average	*#best*	2609.2	2617.2	2790.0	115.1	230.3	1745.3	2764.4	2790.0
$k1$-$k9$	*#opt*	2396.3	2403.5	2503.4	115.1	230.0	1707.0	2480.8	2503.4
	%gap	0.05	0.04	0.01	3.17	1.12	0.58	0.01	0.01

Before discussing the results provided by the tables we recall that lower bound $L_{\lvert\#\leq k_m\rvert}$ is incorporated in bound L_C (see (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 $L_{\lvert\#\leq k_m\rvert}$. This can be clearly observed by looking at the columns

of Tables 3 and 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 does not worsen significantly when started from randomly generated solutions. Looking at the last columns of Tables 3 and 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 those 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*, produced solutions with gaps around $6 \cdot 10^{-3}$ with a maximum of $1.3 \cdot 10^{-2}$ for Class $k3$ (not shown in Table 4), whereas the scatter search always had solutions as close to the lower bound as $1 \cdot 10^{-4}$. On the other hand, the scatter search may require consistently higher computing times.

Heuristic *LPT-k_i* has average gaps close to $3 \cdot 10^{-2}$ for all weight classes. For cardinality classes $k1$–$k6$ its performance improves, while it worsens for $k7$–$k9$. 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 $1 \cdot 10^{-2}$. Heuristic *MS-k_i* has very good performances for almost all classes. 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 determined during the computation of the column generation lower bound L_{CG} (see columns *Scatter* and *Overall*, row $\#opt$), which increased by about 200 units the number of instances solved to optimality.

In Table 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):

- $\%act$ = average percentage of instances where some reduction was obtained;
- $\%n$-*red* = overall percentage of items reduction;
- $\%m$-*red* = overall percentage of subsets reduction;
- *time* = average CPU time.

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.

The reduction phase proved to be an important tool for solving a number of instances to optimality. The corresponding computational effort was however not negligible: By comparing the average CPU times in Table 5 with those in Tables 6 and 7, one can see that the fraction of time taken by the reduction process was about 18% of the total time.

Table 5: Performance of the reduction procedure.

Class	*%act*	*%n-red*	*%m-red*	*time*	*Class*	*%act*	*%n-red*	*%m-red*	*time*
$w1$	20.68	4.33	9.37	1.41	$k1$	0.07	0.01	0.02	0.01
$w2$	19.96	4.26	9.17	1.85	$k2$	0.04	0.01	0.01	0.08
$w3$	20.22	4.27	9.27	1.42	$k3$	15.13	1.46	3.26	13.85
$w4$	21.25	4.47	9.62	1.47	$k4$	0.00	0.00	0.00	0.04
$w5$	20.36	4.30	9.33	1.50	$k5$	0.00	0.00	0.00	0.04
$w6$	20.18	4.32	9.34	1.76	$k6$	0.00	0.00	0.00	0.03
$w7$	20.47	4.19	9.11	1.70	$k7$	66.95	18.12	36.79	0.13
$w8$	19.89	4.30	9.30	1.74	$k8$	53.98	10.78	24.33	0.07
$w9$	20.54	4.34	9.41	1.50	$k9$	47.38	8.40	19.50	0.11
average	20.39	4.31	9.32	1.60	average	20.39	4.31	9.32	1.60

Tables 6 and 7 give more detailed results on the performance of the scatter search algorithm for the selected classes considered in Tables 3 and 4. For each feasible pair (n, m), the entries in Table 6 (resp. Table 7) give 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 and 4;
- t_{1-2} = average computing time for Steps 1 and 2 (inizialization and scatter search);
- t = average total computing time;
- $t_{\max}$ = maximum total computing time.

The results in these tables confirm the robustness and stability of the algorithm, with respect to both variations of weight and cardinality. Instances with up to 400 items and 10 subsets are almost systematically solved to optimality. This behavior worsens 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 12 minutes 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.

We finally examined the impact of scatter search over the performance of the complete algorithm. To this end, the algorithm was also run on our test bed of 25 110 instances

Table 6: Results for selected weight classes

		$w2$					$w5$					$w8$				
n	m	$\%opt$	$\%gap$	t_{1-2}	t	$t_{\max}$	$\%opt$	$\%gap$	t_{1-2}	t	$t_{\max}$	$\%opt$	$\%gap$	t_{1-2}	t	$t_{\max}$
10	3	100.00	0.00	0.03	0.04	0.28	100.00	0.00	0.02	0.03	0.23	100.00	0.00	0.02	0.03	0.28
25	3	98.89	0.00	0.01	0.02	1.04	98.89	0.00	0.01	0.06	2.58	98.89	0.00	0.01	0.06	2.03
50	3	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.05	98.89	0.00	0.01	0.05	4.39
100	3	98.89	0.00	0.03	0.69	60.80	100.00	0.00	0.02	0.02	0.39	100.00	0.00	0.01	0.01	0.06
200	3	100.00	0.00	0.04	0.04	1.26	100.00	0.00	0.03	0.03	0.50	100.00	0.00	0.03	0.03	0.11
400	3	100.00	0.00	0.10	0.10	0.27	100.00	0.00	0.08	0.08	0.28	100.00	0.00	0.11	0.11	0.50
10	4	100.00	0.00	0.02	0.02	0.22	100.00	0.00	0.02	0.02	0.27	100.00	0.00	0.01	0.01	0.06
25	4	98.89	0.00	0.06	0.08	0.99	97.78	0.00	0.04	0.06	1.21	98.89	0.00	0.03	0.04	0.88
50	4	98.89	0.00	0.02	0.07	5.49	97.78	0.00	0.02	0.08	4.01	98.89	0.00	0.02	0.05	3.02
100	4	100.00	0.00	0.02	0.02	0.28	100.00	0.00	0.02	0.02	0.17	100.00	0.00	0.03	0.03	0.44
200	4	100.00	0.00	0.05	0.05	1.81	100.00	0.00	0.04	0.04	0.33	100.00	0.00	0.04	0.04	0.22
400	4	100.00	0.00	0.10	0.10	1.32	100.00	0.00	0.08	0.08	0.55	100.00	0.00	0.08	0.08	0.50
25	5	78.89	0.02	0.27	0.31	1.26	82.22	0.01	0.37	0.41	1.59	85.56	0.01	0.23	0.26	1.11
50	5	98.89	0.00	0.03	0.07	4.34	97.78	0.00	0.04	0.14	6.75	100.00	0.00	0.03	0.03	0.33
100	5	100.00	0.00	0.03	0.03	0.55	98.89	0.00	0.10	0.28	20.65	100.00	0.00	0.05	0.05	0.70
200	5	100.00	0.00	0.08	0.08	1.43	100.00	0.00	0.09	0.09	2.63	100.00	0.00	0.05	0.05	0.17
400	5	100.00	0.00	0.12	0.12	0.77	100.00	0.00	0.12	0.12	0.60	100.00	0.00	0.12	0.12	0.77
25	10	100.00	0.00	0.52	0.53	3.02	100.00	0.00	0.36	0.36	3.07	100.00	0.00	0.45	0.46	2.63
50	10	58.89	0.02	2.12	2.39	7.25	60.00	0.02	2.36	2.78	9.72	71.11	0.01	1.76	2.01	9.94
100	10	100.00	0.00	0.30	0.30	6.15	100.00	0.00	0.32	0.32	4.62	96.67	0.00	0.54	0.92	17.47
200	10	100.00	0.00	0.38	0.38	11.09	100.00	0.00	0.25	0.25	12.14	100.00	0.00	0.10	0.10	0.81
400	10	100.00	0.00	0.99	0.99	70.14	100.00	0.00	0.23	0.23	0.88	100.00	0.00	0.21	0.21	0.98
50	20	92.22	0.01	9.27	9.34	22.19	88.89	0.02	11.12	11.21	71.84	86.67	0.02	11.52	11.61	26.42
100	20	53.33	0.02	8.68	11.80	55.75	55.56	0.02	7.83	12.75	74.70	47.78	0.02	10.27	13.71	74.25
200	20	96.67	0.00	2.07	3.61	96.78	90.00	0.00	3.65	7.53	99.86	92.22	0.00	3.57	7.01	98.42
400	20	100.00	0.00	2.31	2.31	172.30	100.00	0.00	2.35	2.35	173.18	100.00	0.00	2.45	2.45	180.44
100	40	36.67	0.17	66.11	66.80	289.68	37.78	0.17	61.86	62.57	249.19	26.67	0.19	70.65	71.50	223.22
200	40	55.56	0.02	28.93	48.28	159.23	61.11	0.02	27.91	43.46	219.38	58.89	0.02	30.67	50.04	216.41
400	40	94.44	0.00	7.27	10.63	167.20	94.44	0.00	12.43	16.22	603.68	94.44	0.00	17.39	21.78	506.64
200	50	36.67	0.04	55.59	78.27	352.07	27.78	0.05	59.57	84.71	398.87	25.56	0.05	64.81	91.68	408.48
400	50	90.00	0.00	23.18	30.04	474.94	90.00	0.00	18.45	25.04	484.34	92.22	0.00	12.69	17.53	197.80
summary		89.93	0.01	6.73	8.63	474.94	89.64	0.01	6.77	8.75	603.68	89.46	0.01	7.35	9.42	506.64

Table 7: Results for selected cardinality classes

		$k2$					$k5$					$k8$				
n	m	$\%opt$	$\%gap$	t_{1-2}	t	$t_{\max}$	$\%opt$	$\%gap$	t_{1-2}	t	$t_{\max}$	$\%opt$	$\%gap$	t_{1-2}	t	$t_{\max}$
10	3	100.00	0.00	0.01	0.02	0.22	100.00	0.00	0.02	0.02	0.06	100.00	0.00	0.02	0.02	0.06
25	3	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.00	100.00	0.00	0.01	0.03	0.88
50	3	100.00	0.00	0.01	0.01	0.06	100.00	0.00	0.00	0.00	0.06	100.00	0.00	0.00	0.00	0.00
100	3	100.00	0.00	0.02	0.02	0.06	100.00	0.00	0.01	0.01	0.06	100.00	0.00	0.00	0.00	0.00
200	3	100.00	0.00	0.03	0.03	0.11	100.00	0.00	0.03	0.03	0.11	100.00	0.00	0.00	0.00	0.06
400	3	100.00	0.00	0.13	0.13	0.34	100.00	0.00	0.10	0.10	0.28	100.00	0.00	0.00	0.00	0.11
10	4	100.00	0.00	0.02	0.02	0.27	100.00	0.00	0.01	0.01	0.06	100.00	0.00	0.01	0.01	0.06
25	4	98.89	0.00	0.02	0.02	0.55	100.00	0.00	0.00	0.00	0.11	100.00	0.00	0.05	0.07	0.82
50	4	100.00	0.00	0.01	0.01	0.11	100.00	0.00	0.00	0.00	0.06	98.89	0.00	0.03	0.10	8.90
100	4	100.00	0.00	0.04	0.04	0.44	100.00	0.00	0.02	0.02	0.06	100.00	0.00	0.00	0.00	0.00
200	4	100.00	0.00	0.06	0.06	0.22	100.00	0.00	0.03	0.03	0.11	100.00	0.00	0.00	0.00	0.00
400	4	100.00	0.00	0.17	0.17	0.39	100.00	0.00	0.04	0.04	0.16	100.00	0.00	0.01	0.01	0.22
25	5	87.78	0.02	0.24	0.26	1.14	82.22	0.01	0.23	0.24	1.11	80.00	0.01	0.24	0.26	1.15
50	5	100.00	0.00	0.06	0.06	0.49	100.00	0.00	0.01	0.01	0.33	100.00	0.00	0.00	0.00	0.06
100	5	100.00	0.00	0.09	0.09	0.72	100.00	0.00	0.03	0.03	0.06	100.00	0.00	0.00	0.00	0.05
200	5	100.00	0.00	0.09	0.09	0.66	100.00	0.00	0.04	0.04	0.22	100.00	0.00	0.00	0.00	0.11
400	5	100.00	0.00	0.23	0.23	0.55	100.00	0.00	0.06	0.06	0.28	100.00	0.00	0.02	0.02	0.28
25	10	100.00	0.00	0.35	0.35	2.08	100.00	0.00	0.44	0.45	2.69	100.00	0.00	0.33	0.33	2.52
50	10	45.56	0.02	2.85	3.14	7.36	67.78	0.01	1.87	1.95	6.98	46.67	0.02	2.74	2.98	8.62
100	10	97.78	0.00	1.11	1.38	34.94	98.89	0.00	0.52	0.66	23.52	97.78	0.00	0.16	0.41	17.14
200	10	100.00	0.00	0.54	0.54	10.93	100.00	0.00	0.09	0.09	0.33	100.00	0.00	0.01	0.01	0.17
400	10	100.00	0.00	1.41	1.41	41.96	100.00	0.00	0.14	0.14	0.66	100.00	0.00	0.03	0.03	0.78
50	20	92.22	0.01	10.07	10.15	71.84	92.22	0.01	9.92	9.98	22.97	93.33	0.01	9.24	9.32	18.95
100	20	37.78	0.03	14.82	19.57	83.05	52.22	0.02	10.21	11.26	28.39	51.11	0.02	7.64	13.39	42.13
200	20	86.67	0.00	8.87	14.19	99.86	90.00	0.00	3.61	7.85	89.75	100.00	0.00	0.01	0.01	0.50
400	20	100.00	0.00	12.70	12.70	180.44	100.00	0.00	0.38	0.38	2.30	100.00	0.00	0.02	0.02	0.55
100	40	41.11	0.15	53.52	54.09	93.76	30.00	0.19	63.25	63.86	111.34	32.22	0.18	53.90	54.83	91.83
200	40	28.89	0.03	53.02	90.78	178.30	34.44	0.03	44.71	62.76	132.03	95.56	0.00	3.66	6.39	216.41
400	40	92.22	0.00	20.72	26.16	603.68	86.67	0.00	17.08	25.43	206.31	100.00	0.00	0.01	0.01	0.06
200	50	15.56	0.05	71.55	109.25	176.26	47.78	0.03	35.44	41.32	110.13	3.33	0.07	56.05	103.53	180.38
400	50	85.56	0.00	67.45	78.27	734.56	81.11	0.00	23.58	35.26	195.32	100.00	0.00	0.02	0.02	0.22
summary		87.42	0.01	10.33	13.65	734.56	89.14	0.01	6.83	8.45	206.31	90.29	0.18	4.33	6.19	216.41

without scatter search, by doubling the time limit assigned to the column generation phase, in order to compensate for a worse initial solution. It turned out that scatter search is an essential tool for the solution of these problems: Without it, the percentage of instances solved to optimality decreased from 89.7 to 64.1, the average percentage gap increased from 0.01 to 0.47 and the average CPU time increased from about 9 seconds to about 22 seconds.

Acknowledgements

We thank the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) and the Consiglio Nazionale delle Ricerche (CNR), Italy, for the support given to this project. The computational experiments have been executed at the Laboratory of Operations Research of the University of Bologna (LabOR). Thanks are also due to two anonymous referees for helpful comments.

References

[1] L. Babel, H. Kellerer, and V. Kotov. The k-partitioning problem. *Mathematical Methods of Operations Research*, 47:59–82, 1998.

[2] E.G. Coffman, M.R. Garey, and D.S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978.

[3] E.G. Coffman, 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.

[4] 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.

[5] M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200, 1995.

[6] M. Dell'Amico and S. Martello. Bounds for the cardinality constrained $P||C_{\max}$ problem. *Journal of Scheduling*, 4:123–138, 2001.

[7] S.M. Fatemi-Ghomi and F. Jolai-Ghazvini. A pairwise interchange algorithm for parallel machine scheduling. *Production Planning and Control*, 9:685–689, 1998.

[8] A. Frangioni, E. Necciari, and M. G. Scutellà. A multi-exchange neighborhood for minimum makespan machine scheduling problems. *Journal of Combinatorial Optimization*, 2004 (to appear).

[9] G.Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, 19:312–320, 1998.

[10] F. Glover. Parametric combinations of local job shop rules. In *ONR Research Memorandum no. 117*. GSIA, Carnegie Mellon University, Pittsburgh, Pa, 1963.

[11] F. Glover. A multiphase dual algorithm for the zero-one integer programming problem. *Operation Research*, 13:879–919, 1965.

[12] F. Glover. Heuristic for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[13] F. Glover. Scatter search and path relinking. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 297–316. McGraw Hill, 1999.

[14] 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.

[15] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[16] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of ACM*, 34:144–162, 1987.

[17] 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.

[18] R. Hübsher and F. Glover. Applying tabu search with influential diversification to multiprocessor scheduling. *Computers and Operations Research*, 21:877–889, 1994.

[19] 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.

[20] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Handbooks in Operations Research and Management Science*, pages 445–522. North-Holland, Amsterdam, 1993.

[21] S. Martello and P. Toth. Worst-case analysis of greedy algorithms for the subset-sum problem. *Mathematical Programming*, 28:198–205, 1984.

[22] S. Martello and P. Toth. An exact algorithm for the two-constraint 0-1 knapsack problem. *Operations Research*, 51:826–835, 2003.

[23] 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.

[24] E. Mokotoff. Parallel machine scheduling problems: a survey. *Asia-Pacific Journal of Operational Research*, 18:193–242, 2001.

[25] E. Mokotoff, J. J. Jimeno, and I. Gutiérrez. List scheduling algorithms to minimize the makespan on identical parallel machines. *TOP*, 9:243–269, 2001.