2

Scatter Search Algorithms for Identical Parallel Machine Scheduling Problems

Manuel Iori[1] and Silvano Martello[2]

[1] DISMI, Università di Modena e Reggio Emilia, Italy. manuel.iori@unimore.it
[2] DEIS, Università di Bologna, Italy. silvano.martello@unibo.it

Summary. We address the *Identical Parallel Machine Scheduling Problem*, one of the most important basic problems in scheduling theory, and some generalizations of it arising from real world situations. We survey the current state of the art for the most performing meta-heuristic algorithms for this class of problems, with special emphasis on recent results obtained through Scatter Search. We present insights in the development of this heuristic technique, and discuss the combinatorial difficulties of the problems through the analysis of extensive computational results.

Key words: Identical Parallel Machine Scheduling, Scatter Search, Local Search.

2.1 Introduction

Given a set of n *jobs* j, each having an associated processing time p_j ($j = 1, \ldots, n$), and a set of m parallel identical *machines* i ($i = 1, \ldots, m$), each of which can process at most one job at a time, the *Identical Parallel Machine Scheduling Problem* calls for the assignment of each job to exactly one machine, so as to minimize the maximum completion time of a job (*makespan*). We assume, as is usual, that $1 < m < n$, and that the processing times are positive integers. The problem is denoted as $P||C_{max}$ in the three-field classification by Graham, Lawler, Lenstra and Rinnooy Kan [23], and is NP-hard in the strong sense. It is one of the most intensively studied problems in combinatorial optimization, since it has considerable theoretical interest and arises as a subproblem in many real world applications.

The problem is related to another famous combinatorial optimization problem. In the *Bin Packing problem* (BPP) we are given n items, each having an associated size p_j ($j = 1, \ldots, n$), and an unlimited number of identical bins of capacity c: we want to assign each item to one bin, without exceeding its capacity, so that the number of bins used is minimized. Hence BPP can be

M. Iori and S. Martello: *Scatter Search Algorithms for Identical Parallel Machine Scheduling Problems*, Studies in Computational Intelligence (SCI) **128**, 41–59 (2008)
www.springerlink.com

seen as a "dual" of $P||C_{max}$ in which the objective is to minimize the number of machines needed not to exceed a prefixed makespan c.

In this survey we review recent Scatter Search algorithms for $P||C_{max}$ and two generalizations of it that arise from real world situations. In the next section we describe the three considered problems. In Section 2.3 we discuss a general framework for applying Scatter Search to identical parallel machine scheduling problems, focusing on the most effective components of this meta-heuristic technique, and detail its use for these problems. The inherent combinatorial difficulty of the problems and the performance of the Scatter Search algorithms is analyzed in Section 2.4 through the results of extensive computational experiments.

2.2 The problems

We first introduce the basic $P||C_{max}$ problem, and then derive its two generalizations.

2.2.1 Identical Parallel Machine Scheduling Problem

The problem $P||C_{max}$ can be formally stated as an Integer Linear Programming model by introducing a binary variable x_{ij}, taking value one if and only if job j is assigned to machine i ($j = 1, \ldots, n$; $i = 1, \ldots, m$):

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

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

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

$$x_{ij} \in \{0, 1\} \quad (i = 1, \ldots, m; j = 1, \ldots, n), \tag{2.4}$$

where z is the optimum makespan value.

Although this model can be strengthened by means of valid inequalities (as shown in Mokotoff [30]), its computational behavior is unsatisfactory from a practical point of view, due to the high number of variables that take a fractional value in the solution of its linear programming relaxation.

Exact algorithms with better computational behavior were obtained from implicit enumeration techniques. Dell'Amico and Martello [11] proposed combinatorial lower bounds, dominance criteria and a depth-first branch-and-bound algorithm based on the following enumeration scheme. The jobs are sorted so that

$$p_1 \geq p_2 \geq \cdots \geq p_n, \tag{2.5}$$

and are assigned to machines by increasing index. Let $\widetilde{z}$ denote the incumbent solution value, and c_i $(i = 1, \ldots, m)$ the sum of the processing times currently assigned to machine i. At level j of the branch-decision tree, at most m nodes are generated by assigning job j to those machines i such that $c_i + p_j < \widetilde{z}$ (but for each subset of machines having identical c_i value, only the one with minimum index is considered). Computational experiments (see Dell'Amico and Martello [11,13]) showed a good practical behavior, definitely better than the one of the algorithm in [30].

Further improvements were obtained by Dell'Amico, Iori, Martello and Monaci [10] through an algorithm that exploits the dual relationship with the BPP. Given lower and upper bound L and U $(L < U)$ on the optimal solution value, the algorithm performs a binary search which, at each iteration, checks wether there exists a solution with makespan at most $c = \lfloor (L + U)/2 \rfloor$ (and updates L or U correspondingly). The core problem is the decision version of a BPP, which is formulated as a set covering problem and solved through LP relaxation, column generation and branch-and-price.

Computational evidence shows however that all exact algorithms fail when addressing large-size instances, hence the problem has received a large attention from the heuristic point of view. Among the recent relevant contributions we cite the heuristic algorithm based on a simple exchange neighborhood by França, Gendreau, Laporte and Müller [16], the multiple exchanges neighborhoods studied by Frangioni, Necciari and Scutellà [17], the Tabu Search approach by Alvim and Ribeiro [1], the multi-start local search method by Haouari, Gharbi and Jemmali [24] and finally the Scatter Search algorithm by Dell'Amico, Iori, Martello and Monaci [10] which is discussed in Section 2.3.1.

2.2.2 Cardinality Constrained Parallel Machine Scheduling Problem

A well-studied generalization of the $P||C_{max}$ arises when the number of jobs that can be assigned to each machine cannot exceed a given integer k. This can be modeled by adding to the previous model the constraint:

$$\sum_{j=1}^{n} x_{ij} \leq k \quad (i = 1, \ldots, m). \tag{2.6}$$

The model defined by (2.1)–(2.4) and (2.6) describes what is known in the literature as the *Cardinality Constrained Parallel Machine Scheduling Problem*, denoted as $P|\# \leq k|C_{max}$. Problem $P||C_{max}$ is thus the special case of $P|\# \leq k|C_{max}$ arising when $k = n - m + 1$ (as there always exists an optimal solution in which each machine processes at least one job).

Possible applications of this problem arise when m machines (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 machine can have a limit k on the number of different types of operation it can perform, coming,

e.g., from the capacity of the cell tool inventory or the number of robot feeders. If it is imposed that all operations of type j ($j = 1, \ldots, n$) have to be performed by the same machine, and p_j is the total time they require, then $\mathrm{P}|\#\leq k|\mathrm{C}_{\max}$ models the problem of performing all operations with minimum makespan. A real world $\mathrm{P}|\#\leq k|\mathrm{C}_{\max}$ case was presented by Hillier and Brandeau [25], who studied a printed circuit board assembly process inspired by an application at Hewlett-Packard. The dual of $\mathrm{P}|\#\leq k|\mathrm{C}_{\max}$ is a generalization of BPP in which a limit k is imposed to the number of items that can be packed into each bin.

Lower and upper bounds for $\mathrm{P}|\#\leq k|\mathrm{C}_{\max}$ were provided by Babel, Kellerer and Kotov [2] and by Dell'Amico and Martello [12], who also proposed a truncated branch-and-bound algorithm. Felinskas [15] developed genetic algorithms and tested them on the real world case proposed by Hillier and Brandeau [25]. The Scatter Search algorithm by Dell'Amico, Iori and Martello [8] is reviewed in Section 2.3.2.

2.2.3 k_i-Partitioning Problem

Another interesting generalization (of both $\mathrm{P}||\mathrm{C}_{\max}$ and $\mathrm{P}|\#\leq k|\mathrm{C}_{\max}$) is the *$k_i$-Partitioning Problem* (*k_i-PP*), introduced by Babel, Kellerer and Kotov [2]. In this case each machine i has a specific cardinality limit k_i ($i = 1, \ldots, m$). The problem is thus modeled by (2.1)–(2.4), and

$$\sum_{j=1}^{n} x_{ij} \leq k_i \quad (i = 1, \ldots, m). \tag{2.7}$$

In the special case where $k_i = k$ for $i = 1, \ldots, m$ the problem coincides with $\mathrm{P}|\#\leq k|\mathrm{C}_{\max}$.

Possible applications of this problem arise again in Flexible Manufacturing System, when the cells are not identical. The dual of *k_i-PP* is a generalization of BPP in which the bins are numbered by consecutive integers, the first m bins have limits k_i ($i = 1, \ldots, m$) on the number of items that can be packed into them, and all other bins have this limit set to one.

Lower bounds, reduction methods, constructive heuristics and a particular lower bound computation based on column generation were developed by Dell'Amico, Iori, Martello and Monaci [9]. The Scatter Search algorithm proposed by the same authors is described in Section 2.3.3.

2.3 Scatter Search

Scatter Search originates from heuristics for integer programming developed by Glover [18] in the Seventies. Several books and surveys on this methodology can be found in the literature. The reader is referred to the recent survey by Martí, Laguna and Glover [28] and to the special issue of *European Journal of*

Operational Research edited by Martí [14]. Scatter Search can be defined as a population based meta-heuristic that operates on a set of "good" solutions, the *reference set* $\mathcal{RS}$, and iteratively creates new solutions by combining subsets of $\mathcal{RS}$. These new solutions are possibly used to periodically update $\mathcal{RS}$ and the final outcome is the best solution obtained during the search process.

Most implementations of Scatter Search algorithms are based on the template formalized for the first time in Glover [19]. Candidate solutions for the reference set are evaluated on the basis of two criteria: quality and diversity. The *quality* of a solution s coincides or is strictly related to its value, denoted in the following as $z(s)$, while its *diversity* is a relative measure indicating how much its structure differs from that of the solutions that are currently in the reference set. Since the new solutions are created by combining solutions of $\mathcal{RS}$, diversity is a crucial tool for giving a Scatter Search algorithm the possibility of continuously diversifying the search to efficiently explore the solution space.

A concise formulation of the template is as follows:

1. generate a starting population $\mathcal{P}$ of good solutions;
2. create the initial *reference set* $\mathcal{RS}$ by selecting from $\mathcal{P}$ a number of solutions on the basis of their quality and their diversity;
3. **while** a stopping criterion is not met **do**
   3.1 generate a family of subsets of solutions from $\mathcal{RS}$;
   3.2 **for each** subset S of the family **do**
      3.2.1 combine the solutions in S so as to obtain a set of new solutions;
      3.2.2 improve each new solution;
      3.2.3 add to $\mathcal{RS}$ the solutions that meet a quality or diversity criterion
      end for
   end while.

Five main issues have to be dealt with, when developing a Scatter Search algorithm:

A. *Starting population* (Step 1.). The population should consist of high quality solutions that differ consistently from one another. It can be constructed in a totally random way or using heuristics. It is generally convenient to apply an improvement algorithm to each such solution. For the scheduling problems we are considering, Scatter Search appears to be quite robust with respect to the initial population, provided that enough CPU time is given to the overall algorithm.
B. *Improvement* (Steps 1. and 3.2.2). Local search and post optimization are frequently used to improve the solutions quality. Similarly to what happens in the intensification phase of other meta-heuristic approaches, the balance between computational effort and effectiveness may affect dramatically (in positive or in negative) the behavior of the overall algorithm.
C. *Reference set update* (Steps 2. and 3.2.3). The reference set is typically the union of a set Q of high quality solutions and a set D of solutions

highly different from those in Q and from one another. The reference set is usually quite small, typically containing 20 to 30 solutions. Two updating methods are common. A *dynamic* method updates $\mathcal{RS}$ as soon as a solution meets the required quality or diversity criterion, while in a *static* method updating only occurs when all generated subsets have been handled.

D. *Subset generation* (Step 3.1). The method adopted to select a subset of solutions to be combined together can have a relevant effect on the computational effort spent at each iteration of Step 3. Selecting a high number of subsets generally produces, at Step 3.2.1, a high number of new candidate solutions, hence a deep exploration of the current reference set.

E. *Solution combination* (Step 3.2.1). For each subset of solutions, one or more new candidate solutions are produced through combination. This is frequently a crucial aspect of Scatter Search. Specifically tailoring the combination method to the problem can considerably improve the convergence to high quality solutions.

The initial generation method, the improvement algorithms and the solution combination method developed for the three problems we are considering were specifically tailored to the problems, hence are discussed in the next sections. For the reference set update and for the subset generation instead, although several approaches were attempted, it turned out that the most efficient choices were common to the three cases.

Concerning the reference set update (issue C. above), the dynamic method was always used, with different values for $|Q|$ and $|D|$.

The subset generation method (issue D. above) always followed the classical approach proposed by Martí, Laguna and Glover [28], which consists in generating: (i) all the two-solution subsets; (ii) the three-solution subsets obtained by adding to each two-solution subset the solution of highest quality not already contained in it; (iii) the four-solution subsets obtained by the three-solution subsets in the same way as for (ii); (iv) the $|\mathcal{RS}| - 4$ subsets containing the best s solutions, for $s = 5, 6, \ldots, |\mathcal{RS}|$.

In the next sections we detail the specific methods adopted, for issues A., B., and E. above, in the three considered problems.

2.3.1 Scatter Search for $\mathbf{P||C_{max}}$

In the scatter search algorithm proposed by Dell'Amico, Iori, Martello and Monaci [10], a starting population of $|\mathcal{P}| = 40$ solutions was generated through a number of approximation algorithms (see Mokotoff [29], Brucker [3], Leung [27] and Chen [5] for recent surveys on the huge literature on heuristics for $P||C_{max}$). The algorithms used were in particular the well-known *Longest Processing Time* algorithm by Graham [21,22], algorithm *Multi-Subset*, a two-phase approach by Dell'Amico and Martello [11] and other two-phase algorithms proposed by Mokotoff, Jimeno and Gutiérrez [31].

The improvement method adopted was a $k-\ell$ *swap* procedure, used to swap up to k jobs assigned to a machine with up to ℓ jobs assigned to another one, when this leads to decrease the makespan restricted to such two machines. Extensive computational experiments showed that exchanging single jobs ($k = \ell = 1$) gives modest improvements to the solutions quality, exchanging single jobs and pairs of jobs ($k = \ell = 2$) has the best balance between CPU time and efficacy, while higher values of k and/or ℓ produce limited improvements at the expenses of a considerable running time increase.

The reference set $\mathcal{RS}$ was composed by the $|Q| = 10$ solutions with minimum makespan and by the $|D| = 8$ solutions with highest diversity. The diversity of a solution was measured as follows. Given two jobs j and l ($l > j$) and two solutions $s \notin \mathcal{RS}$ and $t \in \mathcal{RS}$, let

$$\delta_{jl}(s,t) = \begin{cases} 1 & \text{if } j \text{ and } l \text{ are processed on the same machine in } s \\ & \text{and on different machines in } t, \text{ or vice versa;} \\ 0 & \text{otherwise,} \end{cases} \tag{2.8}$$

and define the *diversity* of s with respect to $\mathcal{RS}$ as

$$d(s) = \min_{t \in \mathcal{RS}} \left\{ \sum_{j=1}^{\widetilde{n}-1} \sum_{l=j+1}^{\widetilde{n}} \delta_{jl}(s,t) \right\}, \tag{2.9}$$

where $\widetilde{n} = \min(n, 4m)$ is used to limit the evaluation to the largest (hence, most critical) processing times. An accurate definition of the diversity can help in avoiding useless computations. In this case, if $\mathcal{RS}$ contains a solution $\overline{s}$ equivalent to s (i.e., that can be obtained from s by just permuting the machines) equation (2.8) gives $\delta_{ij}(s,\overline{s}) = 0$ for all i and j, so, in (2.9), $d(s)$ has its minimum (of value 0) for $t = \overline{s}$ and s is not added to $\mathcal{RS}$.

From each subset S generated at Step 3.1, new solutions are generated as follows. For each pair of jobs (j,l) $(j,l \leq \widetilde{n})$, let S_{jl} be the subset of S containing those solutions in which j and l are processed on the same machine, and define the sum of the inverse relative errors of such solutions:

$$\varphi_{jl} = \sum_{s \in S_{jl}} \frac{L}{z(s) - L}\,, \tag{2.10}$$

where L denotes the best lower bound value available. (Observe that φ_{jl} tends to be high when j and l are processed on the same machine in many good solutions.) A pair (j,l) is then selected with probability proportional to φ_{jl}, and assigned to the machine of lowest index i for which the current completion time c_i satisfies $c_i + p_j + p_l \leq L$. If no such machine exists, j and l are assigned to the machine i with minimum c_i. The process is iterated until a complete solution s is obtained.

2.3.2 Scatter Search for $\mathbf{P|\#} \leq k|\mathbf{C_{max}}$

A scatter search algorithm for $P|\# \leq k|C_{\max}$ was proposed by Dell'Amico, Iori and Martello [8]. The starting population was obtained by generating $|\mathcal{P}| = 80$ totally random solutions. Attempts to initialize it through a heuristic algorithm by Babel, Kellerer and Kotov [2] or through adaptations of the Multi-Subset algorithm by Dell'Amico and Martello [11] gave limited improvements.

The improvements were obtained through: (i) the $k - \ell$ swap procedure described in Section 2.3.1 with $k = \ell = 1$ (implying that only shifts of single jobs from one machine to another, and one to one job exchanges between pairs of machines were attempted); (ii) a re-optimizing procedure that iteratively fixes the jobs assigned to one machine, and runs a heuristic to schedule the remaining jobs on $m-1$ machines. Additional specially tailored heuristics were also executed for special instances satisfying $n = mk$ (known as *k-partitioning*, see Babel, Kellerer and Kotov [2]), which are particularly difficult to solve in practice.

The reference set $\mathcal{RS}$ was composed by the $|Q| = 8$ solutions with minimum makespan and by the $|D| = 7$ solutions with highest diversity. The evaluation of the diversity of a solution was implemented in a simpler way with respect to $P||C_{\max}$. Let $y_j(s)$ be the machine job j is assigned to in solution s. Then the *diversity* of s with respect to $\mathcal{RS}$ is

$$d(s) = \min_{t \in \mathcal{RS}} \left| \left\{ j \in \{1, \ldots, \widetilde{n}\} : y_j(s) \neq y_j(t) \right\} \right| \tag{2.11}$$

with $\widetilde{n} = 2m$. Using this definition, the reference set update is much faster than using (2.9), but the risk exists of having equivalent solutions in $\mathcal{RS}$.

The combination method was as follows. Given a subset $S \subseteq \mathcal{RS}$ generated at Step 3.1, let $S(i, j) \subseteq S$ be the set of solutions of S in which job j is assigned to machine i, and define an $m \times n$ matrix F with

$$F_{ij} = \sum_{s \in S(i,j)} \frac{z(s)}{z(s) - L} \tag{2.12}$$

A job-machine pair (i, j) has thus an high F_{ij} value if j is processed on i in many high quality solutions. The meaning of F_{ij} in (2.12) is similar to that of φ_{jl} in (2.10). Three solutions are then created through the following random process. For $j = 1, \ldots, n$, job j is assigned to to machine i with probability $F_{ij} / \sum_{h=1}^{m} F_{hj}$. If this makes machine i to have k jobs assigned, F_{il} is set to zero for $l = 1, \ldots, n$ in order to prevent the selection of i at the next iterations. The solution of minimum makespan among the three new solutions is then improved through the local search procedures mentioned above.

2.3.3 Scatter Search for k_i-PP

The scatter search algorithm by Dell'Amico, Iori, Martello and Monaci [9] starts by randomly generating and improving an initial population of $|\mathcal{P}| = 100$ solutions.

Concerning the improvement, the algorithms adopted for $P|\#\leq k|C_{\max}$ were generalized to the considered case of different k_i values.

The reference set $\mathcal{RS}$ had $|Q| = 10$ solutions with lowest makespan, and $|D| = 8$ solutions with high diversity.

The *diversity* of a solution s with respect to $\mathcal{RS}$ was computed as in (2.11). In this case, the advantage of a quick reference set update was preserved with a very limited risk (with respect to $P|\#\leq k|C_{\max}$) of having equivalent solutions in $\mathcal{RS}$, as in k_i-PP the machines are not identical, due to the different k_i values. Also the combination method was very similar to the one adopted for $P|\#\leq k|C_{\max}$, the only difference being in the number of random solutions generated.

2.4 Computational Results

We give here a concise exposition of the computational results obtained by the Scatter Search algorithms described in Section 2.3. In Section 2.4.1 we summarize the main results presented in [8–10] on the three problems addressed, comparing the Scatter Search algorithms with greedy heuristics, local search and exact methods. In Section 2.4.2 we concentrate on $P||C_{\max}$, and compare the Scatter Search algorithm with the most successful meta-heuristics in the literature. In Section 2.4.3 we evaluate the behavior of the three Scatter Search algorithms on the same set of $P||C_{\max}$ instances (remind that the two other problems are generalizations of $P||C_{\max}$). We conclude with Section 2.4.4, where we give some details on the parameters tuning.

2.4.1 Scatter Search vs simple heuristics and exact algorithms

We compare in this section the performance of the three Scatter Search algorithms introduced in Section 2.3 with that of the most well-known greedy heuristics for these three parallel machine scheduling problems, and with three exact approaches consisting in executing, after the Scatter Search, an implicit enumeration algorithm. The heuristics can be classified into three categories: List Scheduling, Threshold and Mixed.

List Scheduling Heuristics initially sort the jobs according to a prespecified criterion, and then assign them to the machines, one at a time, following a given rule. For $P||C_{\max}$ the most famous heuristic of this kind is the *Longest Processing Time* (LPT) by Graham [21], which sorts the jobs according to non-increasing processing time, and then iteratively assigns the next job to the machine having the minimum current completion time. It is known from

the probabilistic analysis by Coffman, Lueker and Rinnooy Kan [7] that, if certain conditions on the processing times are satisfied, the solution produced by LPT is asymptotically optimal. In [12] and [9], LPT was generalized to $P|\#\leq k|C_{\max}$ and k_i-*PP*, respectively, by imbedding in it the cardinality constraints.

Threshold Heuristics exploit the "duality" with BPP defined in Section 2.1. These algorithms consider a tentative value c for the optimum makespan, and solve the corresponding BPP instance, with bin capacity c, using one or more BPP heuristics. If not all items are assigned to the m bins, the remaining items are added through a greedy method. The process is possibly iterated by adjusting the threshold value on the basis of the solution obtained in the current attempt. Examples of threshold algorithms for $P||C_{\max}$ are the *Multi-Subset* (*MS*) method by Dell'Amico and Martello [11], the *Multi Fit* algorithm (*MF*) by Coffman, Garey and Johnson [6], and the ϵ-dual method by Hochbaum and Shmoys [26]. Algorithm MS was adapted to $P|\#\leq k|C_{\max}$ and k_i-*PP*, respectively in [12] and [9].

Mixed Heuristics combine list scheduling and threshold techniques, by switching from one to another during the construction process. Several mixed heuristics were proposed by Mokotoff, Jimeno and Gutiérrez [31] for $P||C_{\max}$, and generalized to k_i-*PP* in [9].

In Tables 2.1–2.3 we compare the results given by the heuristics above and by the Scatter Search. The last column of each table shows the improvement that was obtained by executing an exact algorithm on the instances that had not been solved to optimality by the Scatter Search. For each algorithm we give the percentage of cases where the algorithm found the best solution with respect to the other algorithms (%*best*), the percentage of cases where the algorithm found the optimal solution (%*opt*) and the percentage gap between the solution found by the algorithm and the best lower bound (%*gap*).

In Table 2.1 we compare the results obtained on $P||C_{\max}$ instances by the Scatter Search algorithm of Section 2.3.1 and by heuristics from the literature. The table summarizes the results obtained on two classes of classical $P||C_{\max}$ benchmarks: *uniform instances*, proposed by França, Gendreau, Laporte and Müller [16], and *non-uniform instances*, proposed by Frangioni, Necciari and Scutellà [17]. For a given range $[a, b]$, in the former class each processing time

Table 2.1: Overall performance of heuristics, Scatter Search, and Scatter Search followed by branch-and-price on 780 $P||C_{\max}$ instances.

	List Scheduling	Mixed Heuristics	Threshold Heuristics	Scatter Search	Scatter Search + Branch-and-Price
%*best*	21.9	30.9	79.1	97.7	100.0
%*opt*	21.9	30.9	79.1	97.7	100.0
%*gap*	0.54	0.36	0.05	0.01	0.00

is uniformly randomly generated in $[a, b]$, while in the latter class 98% of the processing times are uniformly randomly generated in $[0.9(b-a), b]$ and the remaining ones in $[a, 0.2(b-a)]$. The considered values were, for both classes, $a = 1$ and $b \in \{100, 1000, 10000\}$. The test instances were obtained by considering all pairs (m, n), with $m \in \{5, 10, 25\}$, $n \in \{10, 50, 100, 500, 1000\}$ and $m < n$, and generating ten instances per pair, resulting in a total of 780 instances[3]).

The experiments were performed on a Pentium IV 3 GHz. The heuristics usually needed less than one CPU second. The Scatter Search was allowed a maximum time limit of 30 seconds for $n \leq 50$, and of 120 seconds for $n > 50$. The branch-and-price algorithm never exceeded one hour CPU time. The table shows that the Scatter Search clearly outperforms all the classical heuristics with respect to the percentages of best and optimal solutions and to the percentage gap from the best lower bound. The last column shows that the improvement obtained by executing the exact branch-and-price algorithm (see [10]) after the Scatter Search is limited, but it allows to solve all instances to optimality.

In Table 2.2 we compare heuristics for $\mathrm{P}|\#\leq k|\mathrm{C}_{\max}$ with the Scatter Search algorithm of Section 2.3.2. The experiments were performed on a large set of fifteen classes of randomly generated instances (twelve proposed by Dell'Amico and Martello [12], and three added by Dell'Amico, Iori and Martello [8]). The first nine classes were obtained by generating the p_j values according to, respectively: (i) uniform distributions in $[10, 1000]$, $[200, 1000]$ and $[500, 1000]$ (Classes 1-3); (ii) exponential distributions with average value $\mu = 25$, 50, 100 (Classes 4-6); (iii) normal distributions with average value $\mu = 100$ and standard deviation $\sigma = 33$, 66, 100 (Classes 7-9). The other six classes were generated as $k-$partitioning instances (with $n = k\,m$): (iv) in Classes 10-12 the p_j values were uniformly distributed in $[500, 10000]$, $[1000, 10000]$ and $[1500, 10000]$; (v) in the additional Classes 13-15 they were generated as "perfect packing" instances (i.e., instances for which the optimal solution value z satisfies $z = \sum_{j=1}^{n} p_j/m$), with $z = 1000$, 5000, 10000. For each class, different instances were created, involving up to 50 machines and 400 jobs, and having cardinality limit values up to 50. In total, 9420 instances were generated.

The algorithms were run on a Pentium IV 2.4 GHz. For most of the instances the complete execution of all the algorithms (including branch-and-bound) needed few seconds in total, and it never exceeded 4 CPU minutes. The Scatter Search was halted if during the last iteration no new solution entered the reference set, or after a maximum of 10 iterations of Step 3. (see the template of Section 2.3). In this case too the Scatter Search clearly outperforms the classical heuristics in terms of percentage of best and optimal solutions found and percentage gap. Running a branch-and-bound algorithm

[3] http://www.inf.puc-rio.br/∼alvim/adriana/tese.html

after the Scatter Search only leads to an improvement of 0.5% in %*best*, and of 0.2% in %*opt*.

In Table 2.3 we finally compare the Scatter Search algorithm of Section 2.3.3 with classical heuristics for k_i-*PP*. The algorithms were run on 81 classes of test problems obtained by combining in all possible ways nine weight classes for the generation of the p_j values and nine cardinality classes for the generation of the k_i values. The weight classes were the first nine classes adopted in [12] for $P|\#\leq k|C_{max}$, and described above. The first six cardinality classes were characterized by a limited range of cardinality limits, obtained by uniformly randomly generating the k_i values so as to ensure $\lceil n/m\rceil - 1 \leq k_i \leq \lceil n/m\rceil + 3$ for $i = 1,\ldots,m$. The last three cardinality classes were characterized by more sparse k_i values, obtained through a particular generation procedure (see [9] for a more detailed description). The algorithms were tested on instances with up to 400 jobs and 50 machines, for a total of 25110 test problems.

The computational experiments were performed on a Pentium III 1.133 GHz. The Scatter Search was halted if either (i) no reference set update occurred during the last iteration, or (ii) Step 3. was executed α times (with $\alpha = 10$ for $n < 100$, $\alpha = 5$ for $100 \leq n < 400$ and $\alpha = 1$ for $n \geq 400$). The algorithms usually needed no more than 10 CPU seconds in total, although this limit could occasionally increase to 500 CPU seconds for particularly difficult instances. The considerations seen for $P|\#\leq k|C_{max}$ apply to this case too. The Scatter Search consistently improves the behavior of the classical heuristics. Adding an exact approach (in this case, a column generation algorithm) only leads to small improvements in the solution quality.

2.4.2 Comparison of meta-heuristic algorithms for $P||C_{max}$

In this section we compare the performance of the Scatter Search algorithm for $P||C_{max}$ (see Section 2.3.1) with that of other meta-heuristic algorithms. The reason for restricting our attention to problem $P||C_{max}$ is that this is the only problem, among the three considered here, for which a clear computational comparison with other meta-heuristics from the literature is possible.

According to our knowledge, the other most successful meta-heuristic algorithms for $P||C_{max}$ are the Tabu Search approach by Alvim and Ribeiro [1]

Table 2.2: Overall performance of heuristics, Scatter Search and Scatter Search followed by branch-and-bound on 9420 $P|\#\leq k|C_{max}$ instances.

	List Scheduling	Threshold Heuristics	Scatter Search	Scatter Search + Branch-and-Bound
%*best*	20.2	32.5	99.5	100.0
%*opt*	20.1	32.5	82.2	82.4
%*gap*	1.65	6.40	0.05	0.05

Table 2.3: Overall performance of heuristics, Scatter Search and Scatter Search followed by column generation on 25110 k_i-PP instances.

	List Scheduling	Mixed Heuristics	Threshold Heuristics	Scatter Search	Scatter Search + Column Generation
$\%best$	4.1	8.3	62.6	99.1	100.0
$\%opt$	4.1	8.2	61.2	88.9	89.7
$\%gap$	3.17	1.12	28.5	0.01	0.01

and the multi-start local search method by Haouari, Gharbi and Jemmali [24]. The algorithm in [1] operates by iteratively defining a tentative threshold for the optimum makespan and solving the associated bin packing problem (see Section 2.1) through a specialized Tabu Search. The algorithm proposed in [24] is based on iterated solutions of a subset-sum problem for assigning jobs to one machine at a time. (Given a set S of n integers, the *subset-sum problem* is to find a subset $S' \subseteq S$ such that the sum of the values in S' is closest to, without exceeding, a given integer c.)

Other interesting results in the meta-heuristic field are the simple exchange neighborhood by França, Gendreau, Laporte and Müller [16], and the multiple exchange neighborhoods studied by Frangioni, Necciari and Scutellà [17]. We do not refer explicitly to these results, as the quality of the solutions they provide is generally worse than that given by the meta-heuristics above.

In Tables 2.4 and 2.5 we compare the three meta-heuristics on the uniform and non-uniform benchmark instances for $P||C_{max}$ described in Section 2.4.1. The entries give: (i) the average percentage gap between the solution value found and the best lower bound ($\%gap$); (ii) the average CPU time spent (sec); (iii) the total number of optimal solutions found ($\#opt$). Each entry refers to 50 instances for $m = 5$ (10 instances each for $n = 10$, 50, 100, 500, 1000) and to 40 instances for $m > 5$ (10 instances each for $n = 50$, 100, 500, 1000). We also give the average and total values over the 130 instances of each range, and over the complete set of 390 instances.

The algorithm by Alvim and Ribeiro [1] was run on a Pentium II 400 MHz with 256 MB RAM. The algorithm by Haouari, Gharbi and Jemmali [24] was run on a Pentium IV 3.2 GHz with 1.5 GB RAM, and the $\%gap$ values were evaluated with respect to a better lower bound than the one used for the two other algorithms. The Scatter Search algorithm by Dell'Amico, Iori, Martello and Monaci [10] was run on a Pentium IV 3 GHz with 512 MB RAM.

The two competitors are faster than the Scatter Search, especially for uniform instances (although all CPU times are very reasonable), but the solutions provided by the Scatter Search are clearly better: the number of optimal solutions found is considerably higher and the percentage gap lower.

2.4.3 Comparison among Scatter Search algorithms

Since both $P|\#\leq k|C_{max}$ and k_i-*PP* are generalizations of $P||C_{max}$, the three Scatter Search algorithms we considered can all be executed on $P||C_{max}$ instances. In order to evaluate the relative merits of the general structure of these algorithms (which is quite similar for the three cases) and of the different specializations adopted to handle the specific constraints, we performed a new series of experiments by executing the Scatter Search algorithms for $P|\#\leq k|C_{max}$ and k_i-*PP* on the same instances used in Table 2.1 for the evaluation of the $P||C_{max}$ Scatter Search. The results obtained are presented in Tables 2.6 and 2.7, as average values on a total of 390 instances per table. The information provided is the same as in the tables of the previous section.

The results for the $P||C_{max}$ Scatter Search were directly taken from [10]. The results for the two other Scatter Search algorithms were obtained by running the corresponding codes on the same computer used in [10], namely a Pentium IV 3 GHz, without varying the termination criteria adopted as best choices for the particular problems addressed.

The tables highlight the relevance of the specializations induced in the algorithms by the cardinality constraints. In Table 2.6 the Scatter Search for $P||C_{max}$ clearly outperforms the ones for $P|\#\leq k|C_{max}$ and k_i-*PP* in terms of solution quality (19 not-optimal solutions versus, respectively, 30 and 29), although it needs a larger CPU time. All the percentage gaps are very small. In Table 2.7 the differences in the solution quality become more evident, since

Table 2.4: Meta-heuristic results on 390 $P||C_{max}$ uniform instances.

		Alvim and Ribeiro			Haouari et al.			Scatter Search		
Range	m	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*
$[1, 10^2]$	5	0.0000	0.00	50	0.0000	0.02	50	0.0000	0.00	50
	10	0.0000	0.00	40	0.0000	0.03	40	0.0000	0.00	40
	25	0.0227	0.00	39	0.0453	0.04	38	0.0227	0.75	39
Average/Total		0.0070	0.00	129	0.0139	0.03	128	0.0070	0.23	129
$[1, 10^3]$	5	0.0000	0.01	50	0.0000	0.08	50	0.0000	0.03	50
	10	0.0019	0.02	38	0.0000	0.09	40	0.0000	0.04	40
	25	0.0322	0.02	37	0.0897	0.10	32	0.0311	5.86	38
Average/Total		0.0105	0.02	125	0.0276	0.09	122	0.0096	1.82	128
$[1, 10^4]$	5	0.0408	0.04	48	0.0406	0.42	48	0.0334	0.05	49
	10	0.0026	0.08	30	0.0021	0.32	30	0.0004	4.64	36
	25	0.0536	0.17	29	0.0257	0.19	28	0.0479	30.79	29
Average/Total		0.0330	0.09	107	0.0242	0.32	106	0.0277	10.92	114
Overall		0.0168	0.04	361	0.0219	0.15	356	0.0147	4.32	371

Table 2.5: Meta-heuristic results on 390 P||C_{max} non-uniform instances.

		Alvim and Ribeiro			Haouari et al.			Scatter Search		
Range	m	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*
$[1, 10^2]$	5	0.0000	0.01	50	0.0000	0.06	50	0.0000	0.04	50
	10	0.1981	0.13	32	0.0000	0.09	40	0.0000	0.04	40
	25	0.0334	0.20	38	0.0000	0.14	40	0.0000	0.37	40
Average/Total		0.0712	0.11	120	0.0000	0.09	130	0.0000	0.14	130
$[1, 10^3]$	5	0.0000	0.01	50	0.0000	0.15	50	0.0000	0.04	50
	10	0.0000	0.12	40	0.0000	0.19	40	0.0000	0.06	40
	25	0.0335	0.41	38	0.0000	0.62	40	0.0000	0.29	40
Average/Total		0.0103	0.17	128	0.0000	0.31	130	0.0000	0.12	130
$[1, 10^4]$	5	0.0000	0.01	50	0.0000	0.74	50	0.0000	0.05	50
	10	0.0001	0.16	38	0.0002	0.56	37	0.0000	0.05	40
	25	0.0338	1.91	33	0.0007	4.11	30	0.0002	10.95	37
Average/Total		0.0104	0.64	121	0.0003	1.72	117	0.0001	3.40	127
Overall		0.0306	0.31	369	0.0001	0.71	377	0.0000	1.22	387

the Scatter Search for P||C_{max} only misses 3 optimal solutions, against the 19 and 18 missed by the two other Scatter Search algorithms. The largest average CPU times were required by the Scatter Search for k_i-*PP*, while the two other algorithms needed on average about one CPU second. The uniform instances appear to be a more difficult test bed than the non-uniform ones, showing a larger average gap and a smaller total number of optimal solutions.

On the other hand we can observe that the Scatter Search template we adopted appears to be very robust. The Scatter Search algorithms for P|#$\leq k$|C_{max} and k_i-*PP*, when executed on P||C_{max} instances, obtain very good results, not too far from the best performance obtained, as it could be expected, by the Scatter Search specifically tailored for this problem.

2.4.4 Parameters Tuning

We finally comment the parameters tuning within the Scatter Search framework, by particularly focusing on the P||C_{max} algorithm of Section 2.3.1, and reviewing the tuning process with reference to the five main points outlined in Section 2.3.

Starting Population. The size of the initial pool of solutions was set to 40, after having tried the values 30, 50, 60, 70 and 100. The algorithm is very robust with respect to this parameter, i.e., all values produced results of comparable quality.

Improvement. The algorithm is very sensitive to this parameter. Limiting the $k - \ell$ swap to $k = \ell = 1$ lead to a non satisfactory performance in which

Table 2.6: Scatter Search algorithms on 390 P||C_{max} uniform instances.

		Scatter Search for P||C_{max}			Scatter Search for P|#$\leq k$|C_{max}			Scatter Search for k_i-*PP*		
Range	m	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*
$[1, 10^2]$	5	0.0000	0.00	50	0.0000	0.00	50	0.0000	0.00	50
	10	0.0000	0.00	40	0.0000	0.00	40	0.0000	0.00	40
	25	0.0227	0.75	39	0.0225	0.01	39	0.0225	0.02	39
Average/Total		0.0070	0.23	129	0.0069	0.00	129	0.0069	0.01	129
$[1, 10^3]$	5	0.0000	0.03	50	0.0000	0.00	50	0.0000	0.00	50
	10	0.0000	0.04	40	0.0021	0.01	38	0.0021	0.23	38
	25	0.0311	5.86	38	0.0320	0.05	38	0.0320	0.75	38
Average/Total		0.0096	1.82	128	0.0105	0.02	126	0.0105	0.30	126
$[1, 10^4]$	5	0.0334	0.05	49	0.0330	0.01	46	0.0329	0.08	47
	10	0.0004	4.64	36	0.0138	0.05	30	0.0091	1.02	30
	25	0.0479	30.79	29	0.0695	0.39	29	0.0658	5.95	29
Average/Total		0.0277	10.92	114	0.0383	0.14	105	0.0357	2.18	106
Overall		0.0147	4.32	371	0.0186	0.05	360	0.0177	0.83	361

only 734 optimum values out of 780 instances were found. Using $k = \ell = 2$ produced the final result (758 optima). Further enlarging the local search with $k = \ell = 3$ very slightly decreased the percentage gap without finding new optima, but required a consistently higher CPU time.

Reference Set Update. Updating the reference set as soon as a solution with high quality or high diversity is found lead to better solutions than updating it at the end of the iteration. Varying the sizes of the two subsets Q and D in the ranges $8 \leq |Q| \leq 12$ and $6 \leq |D| \leq 10$ did not produce relevant variations.

Subset Generation Method. The final choice reported in [10] was to adopt the method proposed by Glover, Laguna and Martí [20] (see Section 2.3). Considering a more limited number of subsets lead to slightly worse performances.

Solution combination. For each subset, the policy of obtaining a single new solution by combination proved to be better than that of generating more (2, 3 or 4) new solutions. The diversity function (2.9) outperformed (2.11), since solutions in which the same groups of jobs are assigned to different machines are not identified as identical by the latter function. Similar arguments hold for the combination method, as the one adopted in [10] outperformed the one in [8]. Other methods such as, e.g., using matrix φ (see (2.10)) to choose only one job at a time, lead to slightly worse results. The solution combination method turned out to be very important, since the Scatter Search algorithm is particularly sensitive to this aspect.

Table 2.7: Scatter Search algorithms on 390 P||C_{max} non-uniform instances.

		Scatter Search for P||C_{max}			Scatter Search for P|#$\leq k$|C_{max}			Scatter Search for k_i-*PP*		
Range	m	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*	*%gap*	*sec*	*#opt*
$[1, 10^2]$	5	0.0334	0.05	49	0.0330	0.01	46	0.0329	0.08	47
	10	0.0004	4.64	36	0.0138	0.05	30	0.0091	1.02	30
	25	0.0479	30.79	29	0.0695	0.39	29	0.0658	5.95	29
Average/Total		0.0000	0.14	130	0.0000	1.02	130	0.0000	2.07	130
$[1, 10^3]$	5	0.0000	0.04	50	0.0000	0.01	50	0.0000	0.18	50
	10	0.0000	0.04	40	0.0000	0.17	40	0.0000	1.69	40
	25	0.0000	0.37	40	0.0000	3.14	40	0.0000	4.81	40
Average/Total		0.0000	0.12	130	0.0000	0.52	130	0.0000	2.21	130
$[1, 10^4]$	5	0.0000	0.04	50	0.0000	0.02	50	0.0000	0.36	50
	10	0.0000	0.06	40	0.0000	0.26	40	0.0000	2.70	40
	25	0.0000	0.29	40	0.0000	1.42	40	0.0000	4.03	40
Average/Total		0.0001	3.40	127	0.0006	1.43	111	0.0004	3.96	112
Overall		0.0000	1.22	387	0.0002	0.99	371	0.0001	2.75	372

2.5 Conclusions

We presented a survey on heuristic results for the well-known Identical Parallel Machine Scheduling Problem and for two generalizations of practical relevance, known as the Cardinality Constrained Parallel Machine Scheduling Problem and the k_i-Partitioning Problem. The problems are particularly challenging from the heuristic point of view, since they present very low percentage gaps between the lower bounds and the upper bounds found by the classical heuristics from the literature. Hence the room for improvement is quite limited.

We described three Scatter Search approaches for these problems, highlighting their common components and their differences. We evaluated the behavior of these algorithms by summarizing the results of extensive computational experiments from the literature, and by presenting new results. The Scatter Search algorithms consistently improve on the results found by the classical heuristics. Using different exact methods leads to limited improvements. A test of the three algorithms on the same set of instances shows that the general approach is very robust.

A possible extensions could be to include Scatter Search in frameworks for parallel machine computing, such as, e.g., the one proposed by Cahon, Melab and Talbi [4].

Acknowledgements

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

References

1. A.C.F. Alvim and C.C. Ribeiro. A hybrid bin-packing heuristic to multiprocessor scheduling. In C.C. Ribeiro and S.L. Martins, editors, *Lecture Notes in Computer Science*, volume 3059, pages 1–13. Springer-Verlag, Berlin, 2004.
2. L. Babel, H. Kellerer, and V. Kotov. The k-partitioning problem. *Mathematical Methods of Operations Research*, 47:59–82, 1998.
3. P. Brucker. *Scheduling Algorithms*. Springer-Verlag, New York, 2001.
4. S. Cahon, N. Melab, and E.-G. Talbi. Paradiseo: A framework for the reusable design of parallel and distributed meta-heuristics. *Journal of Heuristics*, 10(3):357–380, 2004.
5. B. Chen. Parallel scheduling for early completion. In J.Y.T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 9, pages 175–184. CRC Press, Boca Raton, FL, 2004.
6. 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.
7. 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.
8. 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.
9. M. Dell'Amico, M. Iori, S. Martello, and M. Monaci. Lower bounds and heuristic algorithms for the k_i-partitioning problem. *European Journal of Operational Research*, 171:725–742, 2006.
10. M. Dell'Amico, M. Iori, S. Martello, and M. Monaci. Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing*, 2007 (to appear).
11. M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200, 1995.
12. M. Dell'Amico and S. Martello. Bounds for the cardinality constrained $P||C_{\max}$ problem. *Journal of Scheduling*, 4:123–138, 2001.
13. M. Dell'Amico and S. Martello. A note on exact algorithms for the identical parallel machine scheduling problem. *European Journal of Operational Research*, 160:576–578, 2005.
14. R. Martí (ed.). *Feature Cluster on Scatter Search Methods for Optimization. European Journal of Operational Research*, 169, 2, 2006.
15. G. Felinskas. *An investigation of heuristic methods and application to optimization of resource constrained project schedules*. PhD thesis, Vytautas Magnus University, Vilnius, Lithuania, 2007.

16. P.M. França, M. Gendreau, G. Laporte, and F.M. Müller. A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Computers & Operations Research*, 21:205–210, 1994.
17. A. Frangioni, E. Necciari, and M. G. Scutellà. A multi-exchange neighborhood for minimum makespan machine scheduling problems. *Journal of Combinatorial Optimization*, 8:195–220, 2004.
18. F. Glover. Heuristic for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.
19. 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-Verlag, 1998.
20. 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.
21. R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
22. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.
23. 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.
24. M. Haouari, A. Gharbi, and M. Jemmali. Tight bounds for the identical parallel machine scheduling problem. *International Transactions in Operational Research*, 13:529–548, 2006.
25. 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.
26. 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.
27. J.Y.T. Leung (ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Boca Raton, FL, 2004.
28. R. Martí, M. Laguna, and F. Glover. Principles of scatter search. *European Journal of Operational Research*, 169:359–372, 2006.
29. E. Mokotoff. Parallel machine scheduling problems: a survey. *Asia-Pacific Journal of Operational Research*, 18:193–242, 2001.
30. E. Mokotoff. An exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research*, 152:758–769, 2004.
31. E. Mokotoff, J. J. Jimeno, and I. Gutiérrez. List scheduling algorithms to minimize the makespan on identical parallel machines. *TOP*, 9:243–269, 2001.