J Math Model Algor (2010) 9:39–51
DOI 10.1007/s10852-009-9120-6

Minimizing the Makespan in Nonpreemptive Parallel Machine Scheduling Problem

Giampiero Chiaselotti · Maria Italia Gualtieri · Paolamaria Pietramala

Received: 28 January 2009 / Accepted: 12 November 2009 / Published online: 27 November 2009

Abstract A new $n \log n$ algorithm for the scheduling problem of n independent jobs on m identical parallel machines with minimum makespan objective is proposed and its worst-case performance ratio is estimated. The algorithm iteratively combines partial solutions that are obtained by partitioning the set of jobs into suitable families of subsets. The computational behavior on three families of instances taken from the literature provided interesting results.

Keywords Optimization · Scheduling · Approximation algorithm

Mathematics Subject Classifications (2000) 68W25 · 90C59

1 Introduction

In this paper, the scheduling problem of n independent jobs on m parallel machines is considered. All processors are assumed to be identical, that is, the time required to execute a job does not depend on the processor used. Each job may be scheduled independently of the others and the execution, once started, cannot be interrupted (nonpreemptive environment). The objective is to minimize the makespan, that is, the total time required to complete all jobs. Following the standard three field scheduling problems classification scheme of Graham et al. [13], this problem is denoted as $P||C_{max}$.

G. Chiaselotti · M. I. Gualtieri · P. Pietramala (✉)
Dipartimento di Matematica, Università della Calabria,
87036 Arcavacata di Rende, Cosenza, Italy
e-mail: pietramala@unical.it

G. Chiaselotti
e-mail: chiaselotti@unical.it

M. I. Gualtieri
e-mail: mig.gualtieri@unical.it

Despite the simplicity of the statement, the problem is hard to solve: it has been proven that $P||C_{max}$ is NP-hard in the strong sense for an arbitrary $m \geq 2$ [10, 18]. For large instances, one needs to rely on good heuristic algorithms to provide near-optimal results. Heuristic algorithms are classified into constructive algorithms and improvement algorithms. The list scheduling family of Graham [11, 12], which includes the Largest Processing Time (*LPT*), and the MultiFit Decreasing (*MFD*) algorithm of Coffman et al. [4], belongs to the first category. Hochbaum and Shmoys [14] presented a polynomial approximation scheme (*PTAS*) that seems to be, in some sense, the best possible. More precisely, for each ε, Hochbaum and Shmoys provided an algorithm that runs in $O((\frac{n}{\varepsilon})^{\frac{1}{\varepsilon^2}})$-time and has a relative error at most ε. The *LPT* algorithm runs in $O(nlog(n))$-time and its worst-case ratio is equal to $1 + \frac{m-1}{3m}$, whereas the *MFD* algorithm runs in $O(nlog(n) + tnm)$-time and a worst-case ratio is equal to $1 + \frac{2}{11} + \frac{1}{2^t}$ [9, 19], where t represents the number of times that a bin packing problem is solved by using the Lowest Fit Decreasing algorithm (Coffman, Garey and Johnson). Improvement algorithms have been proposed, for example, by França et al. [6], Anderson et al. [1] and Frangioni et al. [8]. Surveys have been provided by Cheng and Sin [3], Lawler et al. [15] and Chen et al. [2]; an overview of existing results and of recent research areas is presented in the handbook edited by Leung [16].

Paletta and Pietramala [17] provided an approximation algorithm that firstly constructs a set of disjoint partial solutions and then iteratively combines the partial solutions in order to get a feasible solution to the scheduling problem. The initial partial solutions are obtained by partitioning the set of jobs into p families of subsets satisfying suitable properties in such a way that the elements of the related processing times m-set are as close to each other as possible.

In this paper, the algorithm of Paletta and Pietramala is modified by changing both the procedure used to partition the jobs into partial solutions and the rule used for selecting which two partial solutions are to be combined; the proofs of the assertions on the algorithm are simplified, and, furthermore, the same worst-case performance ratio is obtained. Moreover, the algorithm runs in $O(nlog(n))$-time as the *LPT* algorithm.

The organization of this paper is as follows. In Section 2 the definition and the properties of the used partitions of the jobs are introduced. In Section 3 the description of the algorithm is contained. In Section 4 the proof of the assertions on the performance of the algorithm is given. Finally, the computational results obtained on three families of instances taken from the literature are presented in Section 5.

2 Definitions and Preliminary Results

Let $I = \{1, ..., i, ..., n\}$ be the set of n independent jobs, $M = \{1, ..., j, ..., m\}$ be the set of m identical parallel processors and $A = \{t_1, ..., t_i, ..., t_n\}$ be the set of processing times of the jobs.

The set of jobs is partitioned in $m \cdot p$ subsets P^r_j, $r = 1, \ldots, p$, $j = 1, \ldots, m$ in such a way that in the p-family $\mathbb{S} = \{\mathbb{S}^1, \ldots, \mathbb{S}^r, \ldots, \mathbb{S}^p\}$, each $\mathbb{S}^r = \{P^r_1, \ldots, P^r_j, \ldots, P^r_m\}$ represents the rth partial solution of the scheduling problem and in $\mathbb{S}^r$, each P^r_j represents the set of jobs that must be performed by the machine j. Each $\mathbb{S}^r$ has associated the m-set $T^r = \{t^r_1, \ldots, t^r_j, \ldots, t^r_m\}$, where $t^r_j := \sum_{i \in P^r_j} t_i$ represents the total processing time needed to the jth machine to perform all the jobs that are assigned to it, that is, all the jobs belonging to P_j^r. The gap between the maximum and the minimum element of the m-set T^r is denoted by Δ^r.

Definition 2.1 A p-family $\mathbb{S} = \{\mathbb{S}^1, \ldots, \mathbb{S}^r, \ldots, \mathbb{S}^p\}$ is called *almost balanced p-family* if the following properties are satisfied:

(a) the m elements of each T^r, $r = 1, \ldots, p$, are sorted in a nonincreasing order with respect to their size, that is, $t_1^r \geq \ldots \geq t_j^r \geq \ldots \geq t_m^r$;
(b) $t_1^r \leq 2t_m^r$, $r = 1, \ldots, p-1$.

Example 2.2 Consider the instance: $I = \{1, 2, ..., 28\}$, $M = \{1, 2, 3, 4, 5\}$ and $A = \{66, 64, 61, 60, 56, 47, 47, 46, 40, 39, 30, 30, 29, 26, 23, 22, 21, 15, 15, 11, 9, 9, 8, 6, 6, 2, 2, 2\}$.

The 4-family

$$\mathbb{S} = \left\{ \begin{array}{l} \mathbb{S}^1 = \{1, \{5, 22\}, 2, \{4, 28\}, 3\}, \\ \mathbb{S}^2 = \{6, 7, \{10, 23\}, 8, \{9, 24\}\}, \\ \mathbb{S}^3 = \{11, 12, 13, \{15, 25\}, 14\}, \\ \mathbb{S}^4 = \{16, 17, \{20, 21\}, \{19, 26\}, \{18, 27\}\} \end{array} \right\},$$

is an almost balanced 4-family of partial solutions. The processing times 5-sets associated to the partial solutions are $T^1 = \{66, 56+9, 64, 60+2, 61\}$, $T^2 = \{47, 47, 39+8, 46, 40+6\}$, $T^3 = \{30, 30, 29, 23+6, 26\}$ and $T^4 = \{22, 21, 11+9, 15+2, 15+2\}$.

Definition 2.3 Let $\mathbb{S}^r$ and $\mathbb{S}^q$ be two partial solutions in an almost balanced family. The *combination* among $\mathbb{S}^r$ and $\mathbb{S}^q$ is defined as the m-family

$$\mathbb{S}^r \uplus \mathbb{S}^q = \{P_1^r \cup P_m^q, \ldots, P_j^r \cup P_{m-j+1}^q, \ldots, P_m^r \cup P_1^q\}$$

and the *sum* between the associated processing times m-sets T^r and T^q as the (not necessarily ordered) m-set $T^r \oplus T^q = \{t_1^r + t_m^q, \ldots, t_j^r + t_{m-j+1}^q, \ldots, t_m^r + t_1^q\}$.

Note that the partial solution $\mathbb{S}^r \uplus \mathbb{S}^q$ is not related to an almost balanced family, because the elements of $T^r \oplus T^q$ are not necessarily sorted in a decreasing order with respect to their size.

Definition 2.4 Let T^r and T^q be the processing times m-sets of the partial solutions $\mathbb{S}^r$ and $\mathbb{S}^q$ belonging to an almost balanced family. The *ordered sum* between T^r and T^q is defined as the ordered m-set $Ord(T^r \oplus T^q)$ whose elements are the elements of $T^r \oplus T^q$ sorted in a non increasing order with respect to their size and the *ordered combination* among $\mathbb{S}^r$ and $\mathbb{S}^q$ as the m-family $Ord(\mathbb{S}^r \uplus \mathbb{S}^q)$ whose sets are those of $\mathbb{S}^r \uplus \mathbb{S}^q$ sorted so that the jth element of $Ord(T^r \oplus T^q)$ represents the total processing time of the jth job-set of $Ord(\mathbb{S}^r \uplus \mathbb{S}^q)$.

Example 2.5 Let T^1 and T^4 be the processing times sets of the partial solutions $\mathbb{S}^1$ and $\mathbb{S}^4$ related to the almost balanced family of Example 2.2, then

$$\begin{aligned}
T^1 \oplus T^4 &= \{66+17, 65+17, 64+20, 62+21, 61+22\} = \{83, 82, 84, 83, 83\},\\
\mathbb{S}^1 \uplus \mathbb{S}^4 &= \{\{1, 18, 27\}, \{5, 22, 19, 26\}, \{2, 20, 21\}, \{4, 28, 17\}, \{3, 16\}\},\\
Ord(T^1 \oplus T^4) &= \{84, 83, 83, 83, 82\},\\
Ord(\mathbb{S}^1 \uplus \mathbb{S}^4) &= \{\{2, 20, 21\}, \{1, 18, 27\}, \{4, 28, 17\}, \{3, 16\}, \{5, 22, 19, 26\}\}.
\end{aligned}$$

In the next proposition is shown that the gap related to the combined partial solution does not increase with respect to the gaps related to the initial partial solutions.

Proposition 2.6 *Let T^r and T^q be the processing times m-sets of the partial solutions $\mathbb{S}^r$ and $\mathbb{S}^q$ belonging to an almost balanced family of partial solutions. Then $\Delta^{T^r \oplus T^q} \leq \max\{\Delta^r, \Delta^q\}$.*

Proof Let $\max\{(T^r \oplus T^q)\} = t^r_k + t^q_{m-k+1}$, for some k, $1 \leq k \leq m$, and $\min\{(T^r \oplus T^q)\} = t^r_l + t^q_{m-l+1}$, for some l, $1 \leq l \leq m$. Let

$$\Delta^{(T^r \oplus T^q)} = (t^r_k + t^q_{m-k+1}) - (t^r_l + t^q_{m-l+1}) = (t^r_k - t^r_l) + (t^q_{m-k+1} - t^q_{m-l+1}).$$

Since $t^r_k - t^r_l \geq 0$ can or cannot occur, both cases will be examined separately.

If $t^r_k - t^r_l \geq 0$, as an immediate consequence of the property (a) in Definition 2.1 it follows that $t^q_{m-k+1} - t^q_{m-l+1} \leq 0$. From this $\Delta^{T^r \oplus T^q} = (t^r_k - t^r_l) + (t^q_{m-k+1} - t^q_{m-l+1}) \leq t^r_k - t^r_l \leq t^r_1 - t^r_m = \Delta^r$.

If $t^r_k - t^r_l \leq 0$, it follows that $t^q_{m-k+1} - t^q_{m-l+1} \geq 0$. Then $\Delta^{T^r \oplus T^q} \leq t^q_{m-k+1} - t^q_{m-l+1} \leq t^q_1 - t^q_m = \Delta^q$. Consequently, $\Delta^{T^r \oplus T^q} \leq \max\{\Delta^r, \Delta^q\}$. □

Remark 2.7 Starting from an almost balanced family of partial solutions, the ordered combination gives an almost balanced family of partial solutions. In fact, the property (a) is guaranteed by the definition of ordered combination operator. Moreover, let T^r and T^q be the processing times m-sets of the partial solutions $\mathbb{S}^r$ and $\mathbb{S}^q$. Let $\max\{(T^r \oplus T^q)\} = t^r_k + t^q_{m-k+1}$, for some k, $1 \leq k \leq m$, and $\min\{(T^r \oplus T^q)\} = t^r_l + t^q_{m-l+1}$, for some l, $1 \leq l \leq m$. Then, $\max\{(T^r \oplus T^q)\} = t^r_k + t^q_{m-k+1} \leq$ (property (a)) $\leq t^r_1 + t^q_1 \leq$ (property (b)) $\leq 2t^r_m + 2t^q_m \leq$ (property (a)) $\leq 2(t^r_l + t^q_{m-l+1}) = 2\min\{(T^r \oplus T^q)\}$.

3 Algorithms

The proposed *SPS* (Sum Partial Solutions) algorithm firstly partitions the jobs by using a procedure, named *DPS* (Determining Partial Solutions), that obtains an almost balanced p-family $\mathbb{S}$ of initial partial solutions to the scheduling problem. Then, at the iteration j, *SPS* selects two partial solutions and combines them with the ordered combination operator, obtaining a single partial solution. The algorithm continues to iterate until a feasible solution of the scheduling problem is obtained. Both the procedure, used to partition the jobs, and the rule, used for selecting which two partial solutions are to be combined, are designed in order to reduce as much as possible the gap between the maximum and minimum elements of the processing times m-set associated to a partial solution.

The *DPS* procedure firstly orders the jobs so that $t_1 \geq \ldots \geq t_i \geq \ldots \geq t_n$. Then it builds an almost balanced p-family of initial partial solutions by processing the jobs in turn, starting with the job 1. Now, we suppose that the job i must be inserted. Then *DPS* selects a initial partial solution that has the biggest gap between the processing times of the first and the last job-sets, say $\mathbb{S}^q$. If $t_i \leq \Delta^q$, then the job i is inserted in last job-set of $\mathbb{S}^q$ and the job-sets are sorted in a nonincreasing order with respect to their processing times. If $t_i > \Delta^q$, then the job i is inserted in the first job-set of a new partial solution.

The *DPS* procedure can be formally described as follows.

```
DPS Procedure

Initialization

– Order the jobs so that $t_1 \geq \ldots \geq t_i \geq \ldots \geq t_n$. Set $p = 1$ ($p =$ number of initial partial solutions);
– Consider $\mathbb{S}^p = \{P_1^p = \{1\}, P_2^p = \emptyset_2, \ldots, P_m^p = \emptyset_m\}$, $T^p = \{t_1^p = t_1, t_2^p = 0, \ldots, t_m^p = 0\}$;
– Set $\Delta^p = t_1$.

Construction

    For each $i = 2, \ldots, n$
        – Compute a processing times $m$-set with the biggest gap, say $T^r$;
            – If $t_i \leq \Delta^r$, then
                – $t_m^r = t_m^r + t_i$, $P_m^r = P_m^r \cup \{i\}$, sort the elements of the $m-$set $T^r$ so that $t_1^r \geq \ldots \geq t_j^r \geq \ldots \geq t_m^r$ and arrange the family $\mathbb{S}^r$ so that $t_j^r$ is the total time required by the jobs belonging to $P_j^r$;
                – $\Delta^r = t_1^r - t_m^r$;
            – End If $t_i \leq \Delta^r$;
            – If $t_i > \Delta^r$, then
                – $p = p + 1$, $\mathbb{S}^p = \{P_1^p = \{i\}, P_2^p = \emptyset_2, \ldots, P_m^p = \emptyset_m\}$, $T^p = \{t_1^p = t_i, , t_2^p = 0, \ldots, t_m^p = 0\}$;
                – $\Delta^p = t_1^p - t_m^p$;
            – End If $t_i > \Delta^r$;
    End For $i$.
```

Proposition 3.1 *The DPS procedure produces an almost balanced p-family that satisfies*

(1) $t_1^1 \geq \ldots \geq t_m^1 \geq t_1^2 \geq \ldots \geq t_m^{p-1} \geq t_1^p \geq \ldots \geq t_m^p$;
(2) *the first job-set of each partial solution is a singleton;*

(3) t^r_j, $j = 1, \ldots, m$ *and* $r = 1, \ldots, p-1$, *is equal to the sum of at least an* $t_i \in A$ *such that* $t_i \geq \min_{q=1,\ldots,p} \{t^q_1\} = t^p_1$;

(4) $\max\{\Delta^1, \ldots, \Delta^p\} \leq t^p_1$;

Proof The assertions (1)–(4) follow from the construction of the algorithm.

Also, by construction, for $r = 1, ..., p-1$, $\Delta^r < t^p_1$, that is $t^r_1 - t^r_m < t^p_1 \leq t^r_m$. This implies $t^r_1 \leq 2t^r_m$. □

Since the number p of initial partial solutions that the *DPS* procedure returns is involved in our approximation ratio, we give some upper bounds for p.

Proposition 3.2 *We assume that* $t_1 \geq \ldots \geq t_i \geq \ldots \geq t_n$.

(a) *The DPS procedure returns* $p \geq \frac{1}{mt_1} \sum_{i=1,\ldots,n} t_i$ *initial partial solutions.*

(b) *Let* $\alpha := \frac{t_1}{t_n}$. *The DPS procedure returns* $p \geq \frac{n}{\alpha m}$ *initial partial solutions.*

(c) *For all instances such that* $t_1 < 2t_n$, *the DPS procedure returns* $p = \left\lceil \frac{n}{m} \right\rceil$ *initial partial solutions* (*here* $\lceil x \rceil$ *denotes the smallest integer not less than* x).

Proof

Statement (a) From (1) and (2) of Proposition 3.1, it follows that $\sum_{j=1,\ldots,m} t^r_j \leq mt^r_1 \leq mt^1_1 = mt_1$, $r = 1, \ldots, p$. Hence $\sum_{i=1,\ldots,n} t_i = \sum_{r=1,\ldots,p} \sum_{j=1,\ldots,m} t^r_j \leq \sum_{r=1,\ldots,p} mt_1 \leq pmt_1$.

Statement (b) By using (a) we have that

$$p \geq \frac{1}{mt_1} \sum_{i=1,\ldots,n} t_i \geq \frac{nt_n}{mt_1} = \frac{n}{\alpha m}.$$

Statement (c) Since $t_1 - t_n < 2t_n - t_n = t_n$, *DPS* returns an almost balanced p-family of initial partial solutions such that every P^r_j, $r = 1, \ldots, p$ and $j = 1, \ldots, m$, is a singleton. It follows that $z = \left\lceil \frac{n}{m} \right\rceil$.

□

Remark 3.3 The family of partial solutions of Examples 2.2 has been determined using the *DPS* procedure.

Now, Proposition 2.6 ensures that the smaller are the gaps associated with the initial partial solutions in $\mathbb{S}$, the smaller are the gaps associated to the combined partial solutions. Moreover, the smaller is the gap associated with the feasible solution, obtained if the ordered combination operator is used $p-1$ times, and the smaller is the heuristic algorithm error. The *SPS* algorithm, at the iteration j, selects two partial solutions which have the biggest gaps and combines them with the ordered combination operator, obtaining a single partial solution. The algorithm continues to run, exactly $p-1$ times, using the same rule and, at the end, returns a solution of the problem. The *SPS* algorithm can be summarized as follows.

SPS Algorithm

Initialization

– Use the *DPS* procedure to obtain an almost balanced p-family $\mathbb{S}$ of initial partial solutions. If *DPS* returns with only one partial solution, then Stop (the algorithm provides an optimal solution).

Construction

```
For $j = 1, \ldots, p-1$
    – Select two ordered partial solutions belonging to $\mathbb{S}$ with associated biggest
      gaps, say $\mathbb{S}^l$ and $\mathbb{S}^k$;
    – Compute $T^{p+j}$=$Ord(T^l \oplus T^k)$, $\mathbb{S}^{p+j}$ =$Ord(\mathbb{S}^l \uplus \mathbb{S}^k)$ and $\Delta^{p+j} = t_1^{p+j} - t_m^{p+j}$;
    – Set $\mathbb{S}$=$(\mathbb{S}\backslash\{\mathbb{S}^l, \mathbb{S}^k\}) \cup \mathbb{S}^{p+j}$;
End For $j$.
```

It is routine to see that the *SPS* algorithm runs in $O(nlog(n))$-time. This is the running time of the *DPS* procedure.

4 The Performance Ratio of *SPS* Algorithm

Let $C_{max} = t_1^{2p-1}$ denote the makespan of the *SPS* solution and C^*_{max} denote the makespan of the optimal solution. As before, $\Delta^{2p-1} = t_1^{2p-1} - t_m^{2p-1}$ denotes the gap of the m-set T^{2p-1}.

From now on we assume that $t_1 \geq \ldots \geq t_i \geq \ldots \geq t_n$.

Theorem 4.1 *Let $A = \{t_1, ..., t_i, ..., t_n\}$ be the set of processing times of the jobs $I = \{1, ..., i, ..., n\}$. Let $\mathbb{S}$ be an almost balanced p-family of initial partial solutions given by the DPS procedure. Then*

$$\frac{C_{max}}{C^*_{max}} \leq 1 + \frac{m-1}{m} \cdot \frac{1}{p} \quad if \quad p > 1$$

and

$$C_{max} = C^*_{max} \quad if \quad p = 1.$$

Proof If the *DPS* procedure returns only the m-set T^1, we have an optimal solution. In fact, $t_1^1 = t_1 = \max\{t_i\}$ and $C^*_{max} \geq t_1$.

Let $p > 1$. In view of (3) of Proposition 3.1, the number of elements $t_i \in A$ such that $t_i \geq t_1^p$ in $t_1^1, \ldots, t_m^1, \ldots, t_1^{p-1}, \ldots, t_m^{p-1}$, is at least $m(p-1)$. These $m(p-1)$ elements $t_i \geq t_1^p$ joined with t_1^p, provides $m(p-1)+1$ jobs with processing times greater or equal to t_1^p (ignoring all the other jobs). Then $C^*_{max} \geq \left\lceil \frac{m(p-1)+1}{m} \right\rceil t_1^p = \left\lceil (p-1) + \frac{1}{m} \right\rceil t_1^p = pt_1^p$. Moreover, one has $\Delta^{2p-1} \leq$ (Proposition 2.6) $\leq \max\{\Delta^1, \ldots, \Delta^p\} \leq$

((4) of Proposition 3.1) $\leq t_1^p$, and therefore $C^*_{max} \geq p\Delta^{2p-1}$. Since $C^*_{max} \geq t_m^{2p-1} + \frac{1}{m}\Delta^{2p-1}$, it follows that

$$C_{max} = t_m^{2p-1} + \Delta^{2p-1} = t_m^{2p-1} + \Delta^{2p-1} - \frac{1}{m}\Delta^{2p-1} + \frac{1}{m}\Delta^{2p-1} \leq C^*_{max} + (1 - \frac{1}{m})\Delta^{2p-1}.$$

Then $\frac{C_{max}}{C^*_{max}} \leq 1 + \frac{1}{C^*_{max}}(1 - \frac{1}{m})\Delta^{2p-1} \leq 1 + \frac{1}{p\Delta^{2p-1}}(1 - \frac{1}{m})\Delta^{2p-1} = 1 + \frac{m-1}{m} \cdot \frac{1}{p}$. □

Corollary 4.2 *For all instances so that $t_1 < 2t_n$, the SPS algorithm returns a solution with*

$$\frac{C_{max}}{C^*_{max}} \leq 1 + \frac{m-1}{m\left\lceil \frac{n}{m} \right\rceil}.$$

Proof The thesis immediately follows from Proposition 3.2 and Theorem 4.1 □

The following example shows that the worst-case performance ratio of our algorithm cannot be improved.

Example 4.3 We consider the instance: $I = \{1, 2, 3, 4, 5\}$, $M = \{1, 2\}$ and $A = \{3, 3, 2, 2, 2\}$. The optimal solution is obtained when jobs 1 and 2 are performed by machine 1 and jobs 3, 4 and 5 are performed by machine 2 so the makespan is equal to 6.

The *DPS* procedure returns the almost balanced 3-family

$$\mathbb{S} = \{\mathbb{S}^1 = \{\{1\}, \{2\}\}, \mathbb{S}^2 = \{\{3\}, \{4\}\}, \mathbb{S}^3 = \{\{5\}, \emptyset\}\}$$

of initial partial solutions and the associated processing times 2-sets $T^1 = \{3, 3\}$, $T^2 = \{2, 2\}$ and $T^3 = \{2, 0\}$. Altogether, the *SPS* algorithm returns

$$\mathbb{S}^5 = \{\{1, 4, 5\}, \{2, 3\}\} \text{ and } T^5 = \{3 + 2 + 2, 3 + 2\} = \{7, 5\}$$

and the related makespan is equal to 7. Then

$$\frac{7}{6} = \frac{C_{max}}{C^*_{max}} \leq 1 + \frac{1}{2} \times \frac{1}{3}.$$

Remark 4.4 The estimate of the worst-case performance ratio of our algorithm depends on the number p that is a priori unknown. However, from Proposition 3.2 we have, for $p > 1$,

$$\frac{C_{max}}{C^*_{max}} \leq 1 + \frac{m-1}{m} \cdot \frac{1}{p} \leq 1 + \frac{(m-1)t_1}{nt_n},$$

that is, a worst-case performance ratio equal to $1 + \frac{1}{c}$ (if $p > 1$), with $c = \frac{nt_n}{(m-1)t_1}$ a priori determined. Therefore, in the instances in which $\frac{1}{c} \leq \frac{2}{11}$, our algorithm works very well compared to the *LPT* and *MFD* algorithms. This happens, for example, when $n > 6(m-1)\frac{t_1}{t_n}$.

Also, our bound is comparable with $1 + \frac{m-1}{mx}$ (where x is the least number of jobs on any processor), that is the worst-case performance bound given by Coffman and Sethi [5] for the *LPT* approach.

5 Computational Results

The Instances In order to compare the *SPS* heuristic with other algorithms, three families of instances taken from the literature, are used for the computational investigation. These instances have quite a different structure.

In the first two families the number of machines m is $5, 10, 25$, the number of jobs n is $50, 100, 500, 1000$ (for $m = 5$ and $n = 10$ is also tested), and the the integer processing times belong to the intervals [1, 100], [1, 1000] and [1, 10000]. Ten instances are randomly generated for each choice of m, n and of the processing time intervals, for a total of 390 instances within each family.

The two families differ in shape of the distribution of processing times. In the first family (UNIFORM), which was presented by França et al., the processing times are generated by using an uniform distribution.

The generator of the second family (NONUNIFORM), which was presented by Frangioni et al. [7, 8], given an interval $[a, b]$ of the processing times, produces instances where 98% of the processing times are uniformly distributed in the interval $[(b - a)0.9, b]$, while the remaining processing times fall within the interval $[a, (b - a)0.02]$.

Both generators are available from http://www.di.unipi.it/di/groups/optimize/Data/index.html. The third family of instances is obtained from several difficult bin packing instances, which are available at the OR-Library of J.E. Beasley, from http://mscmga.ms.ic.ac.uk/jeb/orlib/binpackinfo.html. In this family, denoted BINPACK, the number of jobs n is $120, 250, 500, 1000$, the processing times are uniformly distributed in $\{20, 100\}$ and the number of machines m is the number of bins in the best known solution of the bin packing instances. Twenty instances are generated for each choice of n, for a total of 80 instances.

Plan of the experimentation The *SPS* algorithm is tested on the three families of instances and compared with the classical *LPT* heuristic of Graham, the composite *3-PHASE* heuristic of França et al. and the improvement *1-SPT* algorithm of Frangioni et al.

The results of the *MPS* algorithm of Paletta and Pietramala are not reported because are comparable to the ones obtained using the *LPT*.

The results reported by Frangioni et al. [7] for the *3-PHASE* algorithm, are utilized in this paper, since the same instances are used in this paper. These results does not include the number of instances solved by optimality.

Results are averaged for a group of 10 instances and are given in terms of the relative error with respect to the lower bound

$$L_2 = \max \left\{ \left\lceil \frac{1}{m} \sum_{i=1,\ldots,n} t_i \right\rceil, t_1, t_m + t_{m+1} \right\},$$

where $t_1 \geq t_2 \geq \ldots \geq t_n$.

In the Tables 1–3, columns *LPT*, *3-PHASE*, *1-SPT* and *SPS*, describe the results of the *LPT* heuristic, of the algorithm of França et al., of the algorithm designed by Frangioni et al. and of the algorithm here proposed. The number of instances in which the algorithms obtain the makespan equal to the lower bound, representing instances solved to optimality, is reported in column opt.

Table 1 Computational results for UNIFORM instances

m	n	$p_j \in [1,100]$					$p_j \in [1,1000]$					$p_j \in [1,10000]$				
		LPT		3-PHASE	SPS		LPT		3-PHASE	SPS		LPT		3-PHASE	SPS	
		gap	opt	gap	gap	opt	gap	opt	gap	gap	opt	gap	opt	gap	gap	opt
5	10	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10
	50	4.23e-03		1.96e-04	0.00e-00	10	4.27e-03		2.48e-04	7.24e-04		5.44e-03		5.44e-04	6.99e-04	
	100	1.46e-03	3	0.00e-00	0.00e-00	10	1.22e-03		6.03e-05	5.09e-05	6	9.84e-04		7.78e-05	7.49e-04	
	500	0.00e-00	10	0.00e-00	0.00e-00	10	2.05e-05	4	0.00e-00	0.00e-00	10	4.49e-05		3.96e-07	0.00e-00	10
	1000	0.00e-00	10	0.00e-00	0.00e-00	10	6.00e-06	4	0.00e-00	0.00e-00	10	1.03e-05		0.00e-00	0.00e-00	10
10	50	1.60e-02	1	2.35e-03	5.63e-03	1	2.80e-02		4.43e-03	5.53e-03		2.07e-02		4.81e-03	7.66e-03	
	100	4.93e-03	2	0.00e-00	0.00e-00	10	4.76e-03		3.43e-04	6.76e-04		5.63e-03		2.48e-04	7.16e-04	
	500	4.10e-05	9	0.00e-00	0.00e-00	10	1.61e-04		0.00e-00	0.00e-00	10	1.59e-04		2.38e-06	1.61e-06	6
	1000	0.00e-00	10	0.00e-00	0.00e-00	10	4.01e-05		0.00e-00	0.00e-00	10	4.81e-05		0.00e-00	0.00e-00	10
25	50	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10
	100	2.52e-02		3.62e-03	1.83e-02	1	2.23e-02		4.04e-03	2.81e-02		3.26e-03		4.33e-03	1.96e-02	
	500	5.04e-04	5	0.00e-00	0.00e-00	10	1.06e-03		0.00e-00	2.96e-05	7	1.31e-03		2.28e-05	5.08e-05	
	1000	1.01e-04	8	0.00e-00	0.00e-00	10	2.81e-04		0.00e-00	0.00e-00	10	3.55e-04		3.52e-06	3.52e-06	5

Table 2 Computational results for NONUNIFORM instances

m	n	$p_j \in [1,100]$					$p_j \in [1,1000]$					$p_j \in [1,10000]$				
		LPT		3-PHASE	SPS		LPT		3-PHASE	SPS		LPT		3-PHASE	SPS	
		gap	opt	gap	gap	opt	gap	opt	gap	gap	opt	gap	opt	gap	gap	opt
5	10	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10
	50	1.69e-02		1.54e-02	1.63e-02		1.75e-02		1.59e-02	1.65e-02		1.76e-02		1.58e-02	1.66e-02	
	100	1.67e-02		6.05e-03	1.57e-02		1.70e-02		6.52e-03	1.60e-02		1.70e-02		6.64e-03	1.60e-02	
	500	5.86e-04		2.03e-03	4.69e-00	2	6.35e-04		0.00e-00	5.25e-04	1	6.42e-04		0.00e-00	5.33e-04	1
	1000	1.44e-04		1.25e-03	2.66e-05	7	1.78e-04		1.88e-04	6.38e-05	5	1.78e-04		0.00e-00	5.60e-05	5
10	50	1.76e-02		1.42e-02	1.67e-02		1.82e-02		1.42e-02	1.70e-02		1.82e-02		1.42e-02	1.72e-02	
	100	1.72e-02		7.31e-03	1.61e-02		1.77e-02		7.18e-03	1.65e-02		1.78e-02		7.25e-03	1.66e-02	
	500	1.00e-02		6.63e-03	9.40e-03		1.01e-02		6.46e-03	9.52e-03		1.02e-02		6.65e-03	9.54e-03	
	1000	3.83e-04	1	2.74e-03	9.57e-05	7	4.12e-04		4.53e-04	9.25e-05	7	4.10e-04		0.00e-00	9.83e-05	6
25	50	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10	0.00e-00	10	0.00e-00	0.00e-00	10
	100	1.77e-02		7.44e-03	1.66e-02		1.81e-02		7.50e-03	1.73e-02		1.81e-02		7.58e-03	1.74e-02	
	500	9.79e-03		6.43e-03	9.25e-03		1.00e-02		6.59e-03	9.44e-03		1.01e-02		6.58e-03	9.46e-03	
	1000	9.30e-03		6.16e-03	8.77e-03		9.38e-03		6.18e-03	8.81e-03		9.40e-03		6.20e-03	8.83e-03	

Table 3 Computational results for BINPACK instances

m	*n*	*LPT*	*1-SPT*	*SPS*
		gap	gap	gap
48	120	1.16e-01	2.30e-02	2.96e-02
50	120	1.33e-01	1.90e-02	3.04e-02
102	250	1.20e-01	2.28e-02	2.54e-02
102	250	1.42e-01	1.94e-02	2.34e-02
203	500	1.26e-01	2.48e-02	2.60e-02
200	500	1.41e-01	1.33e-02	2.93e-02
402	1000	1.32e-01	1.80e-02	2.87e-02
399	1000	1.35e-01	2.67e-02	2.60e-02

All the instances of the three families are solved so quickly by the *SPS* algorithm that the running time in which the makespan is obtained, is not reported. The *SPS* algorithm is coded in *C* and all tests are executed on a Pentium 3 machine.

Computational results The computational results obtained using the *SPS* algorithm are very encouraging.

UNIFORM instances are known to be efficiently approached with the *LPT* and *3-PHASE* algorithms. *LPT* usually obtains low gaps, and solves to optimality a fair number of instances, while *3-PHASE* offers more accurate results than *LPT*. The *SPS* algorithm noticeably improves *LPT*, and in 27 out of 39 cases the average relative error of *SPS* is equal or less than the one given by the more accurate *3-PHASE*. Moreover, when the ratio $\frac{n}{m}$ is large, *SPS* provides an optimal solution almost always. The results for UNIFORM instances are shown in Table 1.

For NONUNIFORM instances the average relative error of *SPS* is always slightly lower than the average relative error of *LPT* and comparable to those of *3-PHASE*, especially when the ratio $\frac{n}{m}$ is large. The results for NONUNIFORM instances are shown in Table 2.

For BINPACK instances the average relative error of *SPS* is always lower than the average relative error of *LPT* and comparable to those of *3-PHASE*. In these instances the ratio $\frac{n}{m}$ is small, but the same happens for the the ratio $\frac{t_1}{t_m}$. The results for BINPACK instances are shown in Table 3.

6 Conclusions

The paper considers the problem of scheduling independent jobs on identical parallel machines with the objective to minimize the makespan, that is the maximum finishing time over all machines. For this problem, a new *nlogn* algorithm is presented, that firstly constructs a set of partial solutions and secondly iteratively combines them to get a feasible solution to the scheduling problem. A worst-case performance ratio is also given that depends on the number p of initial partial solutions.

The presented algorithm is capable of producing good quality solutions for all classes of instances used in the computational investigation. The tests show that the *SPS* algorithm outperforms the *LPT* heuristic and is comparable to the composite *3-PHASE* heuristic.

Acknowledgements The authors wish to thank F.M. Muller and E. Necciari for having provided the generators and the instances used in the computational testing. The authors also wish to thank the Referees for the constructive comments.

References

1. Anderson, E.J., Glass, C.A., Potts, C.N.: Machine scheduling. In: Aarts, Lenstra (eds.) Local Search in Combinatorial Optimization, pp. 361–414. Wiley, New York (1997)
2. Chen, B., Potts, C.N., Woeginger, G.J.: A review of machine scheduling: complexity, algorithms and approximability. In: Du, D.Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, vol. 3, pp. 21–169. Kluwer Academic, Dordrecht (1998)
3. Cheng, T., Sin, C.: A state of the art review of parallel machine scheduling research. Eur. J. Oper. Res. **47**, 271–292 (1990)
4. Coffman, E.G. Jr., Garey, M.R., Johnson, D.S.: An application of bin-packing to multiprocessor scheduling. SIAM J. Comput. **7**, 1–17 (1978)
5. Coffman, E.G. Jr., Sethi, R.: A generalized bound on LPT sequencing. RAIRO-Inform. **10**, 17–25 (1976)
6. França, P.M., Gendrau, M., Laporte, G., Muller, F.M.: A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. Comput. Ops. Res. **21**(2), 205–210 (1994)
7. Frangioni, A., Scutellà, M.G., Necciari, E.: Multi-exchange algorithms for the minimum makespan machine. Technical Report:99-22, Dipartimento di Matematica. University of Pisa, Italy (1999)
8. Frangioni, A., Necciari, E., Scutellà, M.G.: A multi-exchange neighborhood for minimum makespan machine scheduling problems. J. Comb. Optim. **8**, 195–220 (2004)
9. Friesen, D.K.: Tighter bounds for the MultiFit processor scheduling algorithm. SIAM J. Comput. **13**, 170–181 (1984)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: a Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
11. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**, 1563–1581 (1966)
12. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 416–429 (1969)
13. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling a survey. Ann. Discrete Math. **5**, 287–326 (1979)
14. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. Assoc. Comput. Mach. **34**, 144–162 (1987)
15. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and scheduling: algorithms and complexity in logistics of production and inventory. In: Handbooks in Operation Research and Management Science, vol. 4, pp. 445–522. Elsevier Science, Amsterdam (1993)
16. Leung, J. (ed.): Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman & Hall/CRC, Boca Raton (2004)
17. Paletta, G., Pietramala, P.: A new approximation algorithm for the non-preemptive scheduling of independent jobs on identical parallel processors. SIAM J. Discrete Math. **21**, 313–328 (2007)
18. Ullman, J.D.: Complexity of sequencing problems. In: Coffman, E.G. (ed.) Computer and Job Shop Scheduling Theory, pp. 139–164. Wiley, New York (1976)
19. Yue, M.: On the exact upper bound for the MultiFit processor scheduling algorithm. Ann. Oper. Res. **24**, 233–259 (1990)