Int. J. Contemp. Math. Sciences, Vol. 3, 2008, no. 1, 25 - 36

A New $n \log n$ Algorithm for the Identical Parallel Machine Scheduling Problem

Maria Italia Gualtieri

Dipartimento di Matematica, Università della Calabria
87036 Arcavacata di Rende, Cosenza, Italy
mig.gualtieri@unical.it

Giuseppe Paletta

Dipartimento di Economia e Statistica, Università della Calabria
87036 Arcavacata di Rende, Cosenza, Italy
g.paletta@unical.it

Paolamaria Pietramala

Dipartimento di Matematica, Università della Calabria
87036 Arcavacata di Rende, Cosenza, Italy
pietramala@mat.unical.it

Abstract. A new constructive heuristic for the scheduling problem of n independent jobs on m identical parallel machines with minimum makespan objective is described. The proposed algorithm, which is an $n \log n$ algorithm as the LPT algorithm of Graham, iteratively combines partial solutions that are obtained by partitioning the set of jobs in suitable families of subsets. The algorithm was tested using different families of instances which were taken from the literature and results compared with other well known algorithms.

Mathematics Subject Classification: 90C59, 90B35

Keywords: Combinatorial optimization, Scheduling, Heuristic algorithm

1. Introduction

In this paper the scheduling problem of n independent jobs on m parallel machines is considered. Each job i must be processed without interruption by only one of the m machines (non-preemptive environment); as the machines are identical, the processing time p_i of the job i is independent of the processing

machine (environment of identical parallel processors). The objective is to minimize the makespan, i.e. the total time required to complete all jobs. Using the standard three field classification scheme of Graham et al. (1979), this problem is usually denoted as $P||C_{max}$.

The problem is well known to be NP-hard in strong sense for an arbitrary $m \geq 2$, see Garey and Johnson (1978), and Ullman (1976). For large instances, one needs to rely on good heuristic procedures to provide solutions that are probably close to the optimum. Heuristic algorithms are classified into constructive algorithms and improvement algorithms. In the first category, the list scheduling family of Graham (1966 and 1969), which includes the Largest Processing Time (LPT), and the MultiFit Decreasing (MDF) scheduling algorithm of Coffman et al. (1978) can be referred. Improvement algorithms have been proposed by França et al. (1994), Anderson et al. (1997) and Frangioni et al. (2004), among others.

Surveys regarding the heuristic algorithms for parallel machine scheduling problems have been provided by Cheng and Sin (1990), by Lawler et al. (1993) and by Chen et al. (1998).

In this paper, a constructive $n \log n$ algorithm, named PSC, is presented. It is more accurate with respect to the relative error than the $n \log n$ LPT algorithm of Graham. As in Paletta and Pietramala (2007), the new algorithm is based on the idea of combining iteratively partial solutions, calculated by partitioning the set of jobs in suitable families of subsets as described below, until a feasible solution for the scheduling problem is obtained. The algorithm of Paletta and Pietramala is modified by changing both the procedure used to partition the set of jobs into partial solutions and the rule used for selecting which two partial solutions are to be combined.

In order to compare PSC with other algorithms, different families of instances, taken from the literature, were used for the computational investigation.

The paper is organized as follows. Section 2 presents the definitions and the properties of the partitions that are used to design the algorithm. Section 3 contains the description of the algorithm. Finally, Section 4 includes results of the computational investigation concerning $P||C_{max}$.

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 machines and $A = \{p_1, ..., p_i, ..., p_n\}$ be the set of processing times of the jobs.

Let $\mathfrak{I} = \{I_1^1, ..., I_j^1, ..., I_m^1, ..., I_1^r, ..., I_j^r, ..., I_m^r, ..., I_1^z, ..., I_j^z, ..., I_s^z\}$, $s \leq m$, be a partition of the set I.

The family of z partial solutions $\mathfrak{P} = \{\mathfrak{I}^1, \ldots, \mathfrak{I}^r, \ldots, \mathfrak{I}^z\}$ is associated to the partition $\mathfrak{I}$ of I, where $\mathfrak{I}^r = \{I_1^r, \ldots, I_j^r, \ldots, I_m^r\}$, $r = 1, ..., z-1$, represents

the r-th partial solution and $\mathfrak{I}^z = \{I_1^z, \ldots, I_j^z, \ldots, I_s^z, \emptyset_{s+1}, \ldots, \emptyset_m\}$, $s \leq m$, represents the z-th partial solution.

With respect to a partial solution $\mathfrak{I}^r$, I_j^r represents the set of jobs that are performed by the machine j; moreover, in the partial solution $\mathfrak{I}^z$, the symbol $\emptyset_j$, $j = s+1, \ldots, m$, indicates that the machine j is not performing any jobs.

Let $p_j^r := \sum_{i \in I_j^r} p_i$ be the sum of the processing times of the jobs belonging to I_j^r, $r = 1, ..., z-1$ and $j = 1, ..., m$; let $p_j^z := \sum_{i \in I_j^z} p_i$ be the sum of the processing times of the jobs belonging to I_j^z for $j = 1, ..., s$ and $p_j^z := 0$ for $s < j \leq m$. Each partial solution $\mathfrak{I}^r$, $r = 1, ..., z-1$, has, associated with it, the processing times m-set $G^r = \{p_1^r, \ldots, p_j^r, \ldots, p_m^r\}$, whereas $\mathfrak{I}^z$ has, associated with it, the processing times m-set $G^z = \{p_1^z, \ldots, p_s^z, 0, \ldots, 0\}$, $s \leq m$.

Definition 1. A partial solutions $\mathfrak{I}^r$ is called *ordered partial solution* if the elements of G^r are sorted in not increasing order with respect to their size i.e.

$$p_1^r \geq \ldots \geq p_j^r \geq \ldots \geq p_m^r.$$

Definition 2. A family of z partial solutions $\mathfrak{P} = \{\mathfrak{I}^1, \ldots, \mathfrak{I}^r, \ldots, \mathfrak{I}^z\}$ is called *ordered z-family of solutions* if each $\mathfrak{I}^r$ is an *ordered partial solution.*

Definition 3. Let $\mathfrak{I}^r$ and $\mathfrak{I}^q$ be two *ordered partial solutions.* The *"combination"* among $\mathfrak{I}^r$ and $\mathfrak{I}^q$ ($\mathfrak{I}^r \uplus \mathfrak{I}^q$) is defined as the m-family

$$\mathfrak{I}^r \uplus \mathfrak{I}^q = \{I_1^r \cup I_m^q, \ldots, I_j^r \cup I_{m-j+1}^q, \ldots, I_m^r \cup I_1^q\}.$$

Thus $\mathfrak{I}^r \uplus \mathfrak{I}^q$ corresponds to a new partial solution using the jobs in $\mathfrak{I}^r$ and $\mathfrak{I}^q$, and the set $I_j^r \cup I_{m-j+1}^q$, $j = 1, \ldots, m$, represents the jobs performed by the machine j in the new partial solution. The total processing time needed for the machines to perform all the jobs belonging to $\mathfrak{I}^r \uplus \mathfrak{I}^q$ is computed by using the following definition.

Definition 4. Let G^r and G^q be the sets of processing times of the *ordered partial solutions* $\mathfrak{I}^r$ and $\mathfrak{I}^q$. The *"sum"* among G^r and G^q ($G^r \oplus G^q$) is defined as the m-set

$$G^r \oplus G^q = \{p_1^r + p_m^q, \ldots, p_j^r + p_{m-j+1}^q, \ldots, p_m^r + p_1^q\}.$$

Therefore $p_j^r + p_{m-j+1}^q$ represents the total processing time required to perform all the jobs belonging to $I_j^r \cup I_{m-j+1}^q$, and $\mathfrak{I}^r \uplus \mathfrak{I}^q$ is a partial solution that is not necessarily *ordered* because the elements of $G^r \oplus G^q$ are not sorted in decreasing order with respect to their size.

Definition 5. Let G^r and G^q be the sets of processing times of the *ordered partial solutions* $\mathfrak{I}^r$ and $\mathfrak{I}^q$. The *"ordered sum"* among G^r and G^q is defined as the ordered m-set $Ord(G^r \oplus G^q)$ whose elements are the elements of $G^r \oplus G^q$ sorted in non-increasing order with respect to their size.

Definition 6. Let $\mathfrak{I}^r$ and $\mathfrak{I}^q$ be two *ordered partial solutions*. The *"ordered combination"* among $\mathfrak{I}^r$ and $\mathfrak{I}^q$ is defined as the m-family $Ord(\mathfrak{I}^r \uplus \mathfrak{I}^q)$ whose sets are those of $\mathfrak{I}^r \uplus \mathfrak{I}^q$ sorted such that the j-th element of $Ord(G^r \oplus G^q)$ represents the total processing time of the j-th job-set of $Ord(\mathfrak{I}^r \uplus \mathfrak{I}^q)$.

Let $\mathfrak{I}^r$ be an *ordered partial solution*. Let $\Delta^r := p^r_1 - p^r_m$ denote the gap between the maximum and the minimum element of the m-set G^r.

The *sum* operator satisfies the property indicated in the following proposition.

Proposition 1. *Let G^r and G^q be the sets of processing times of the ordered partial solutions $\mathfrak{I}^r$ and $\mathfrak{I}^q$. Put $S = G^r \oplus G^q$ and $\Delta^S = \max\{S\} - \min\{S\}$. Then $\Delta^S \leq \max\{\Delta^r, \Delta^q\}$.*

Proof. Let

$$\max\{S\} = p^r_k + p^q_{m-k+1} \qquad \text{for some } k, \quad 1 \leq k \leq m,$$

and

$$\min\{S\} = p^r_l + p^q_{m-l+1} \qquad \text{for some } l, \quad 1 \leq l \leq m.$$

Let

$$\Delta^S = (p^r_k + p^q_{m-k+1}) - (p^r_l + p^q_{m-l+1}) = (p^r_k - p^r_l) + (p^q_{m-k+1} - p^q_{m-l+1}).$$

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

First case: $p^r_k - p^r_l \geq 0$.
As an immediate consequence of the definitions of S and *ordered partial solution*, it follows that $p^q_{m-k+1} - p^q_{m-l+1} \leq 0$.

¿From this

$$\Delta^S = (p^r_k - p^r_l) + (p^q_{m-k+1} - p^q_{m-l+1}) \leq p^r_k - p^r_l \leq p^r_1 - p^r_m = \Delta^r.$$

Second case: $p^r_k - p^r_l \leq 0$.
As an immediate consequence of the definitions of S and *ordered partial solution*, it follows that $p^q_{m-k+1} - p^q_{m-l+1} \geq 0$.

Then

$$\Delta^S = (p^r_k - p^r_l) + (p^q_{m-k+1} - p^q_{m-l+1}) \leq p^q_{m-k+1} - p^q_{m-l+1} \leq p^q_1 - p^q_m = \Delta^q.$$

Consequently

$$\Delta^S \leq \max\{\Delta^r, \Delta^q\}.$$

□

In this paper, the procedure used to partition the jobs in an *ordered z-family of solutions* $\mathfrak{P} = \{\mathfrak{I}^1, \ldots, \mathfrak{I}^r, \ldots, \mathfrak{I}^z\}$ was designed to reduce as much as possible the gap Δ^r related to $\mathfrak{I}^r$. So, from Proposition 1 it follows that the smaller the gaps associated with initial partial solutions in $\mathfrak{P}$, the smaller are the gaps associated to the *partial solutions* in the *ordered (z-1)-family of solutions*

$$\{Ord(\mathfrak{I}^r \uplus \mathfrak{I}^q), \mathfrak{I}^1, ..., \mathfrak{I}^{r-1}, \mathfrak{I}^{r+1}, \ldots, \mathfrak{I}^{q-1}, \mathfrak{I}^{q+1}, ..., \mathfrak{I}^z\}.$$

If the *ordered combination* operator is used $z-1$ times until a single ordered solution, e.g. $\mathfrak{I}^{2z-1}$, is obtained, an upper bound on the final gap Δ^{2z-1} of the processing times m-set G^{2z-1} can be given by:

Proposition 2. *Let $\mathfrak{P}$ be an ordered z-family of solutions. Then*

$$\Delta^{2z-1} \leq \max_{q=1,\ldots,z} \{\Delta^q\}.$$

3. Algorithms

The proposed *PSC* algorithm partitions the jobs by using a procedure, named *OFS*, for obtaining an *ordered z-family of solutions* to the scheduling problem. Then, at iteration j, *PSC* selects two *ordered partial solutions* (e.g. $\mathfrak{I}^l$ and $\mathfrak{I}^k$) and combines them with the *ordered combination* operator obtaining a single *ordered partial solution* (e.g. $\mathfrak{I}^{z+j}$). The algorithm continues to iterate (exactly $z-1$ times) until the feasible solution $\mathfrak{I}^{2z-1}$ of the scheduling problem is obtained.

The procedure used to partition the jobs in partial solutions and the rule used for selecting which two partial solutions are to be combined are critical to the success of the heuristic algorithm. The partition procedure and the selection rule were designed to reduce as much as possible the gap between the maximum and minimum elements of the processing times m-set, which is associated to the partial solution.

The algorithm can be summarized as follows.

Algorithm *PSC*

Initialisation

- Use the procedure *OFS* to obtain an *ordered z-family of solutions* $\mathfrak{P}$ so that $\Delta^1 \geq \ldots \geq \Delta^r \geq ... \geq \Delta^z$. If *OFS* returns with only one partial solution then Stop (the algorithm returns with an optimal solution);
- When equal Δ^r values are obtained, then sort these Δ^r in non-decreasing order with respect to the sum $\sum_{j=2,\ldots,m} (p_1^r - p_j^r)$.

Construction

For $j = 1, \ldots, z-1$

-select the first two ordered partial solutions belonging to $\mathfrak{P}$ (say $\mathfrak{I}^l$ and $\mathfrak{I}^k$);
-compute G^{z+j}=$Ord(G^l \oplus G^k)$, $\mathfrak{I}^{z+j}$ =$Ord(\mathfrak{I}^l \uplus \mathfrak{I}^k)$ and $\Delta^{z+j} = p_1^{z+j} - p_m^{z+j}$;

-set $\mathfrak{P}$=$(\mathfrak{P}\backslash\{\mathfrak{I}^l, \mathfrak{I}^k\}) \cup \mathfrak{I}^{z+j}$ and order $\mathfrak{P}$ so that the Δ^r values are in non-decreasing order;

End For j.

Let G_j^{2z-1} be the total processing time needed for the j-th machine to perform all the jobs assigned to it by the *PSC* solution. Thus $\Delta = \Delta^{2z-1} = G_1^{2z-1} - G_m^{2z-1}$ represents an upper bound to the heuristic algorithm error. When this difference is equal to zero, an optimal solution is obtained.
Moreover, as Proposition 2 ensures that $\Delta \leq \max\limits_{q=1,\dots,z}\{\Delta^q\}$, the smaller the gaps of initial partial solutions in $\mathfrak{P}$, the smaller is the Δ associated with the feasible solution, and therefore, the smaller is the upper bound to the heuristic algorithm error. In particular, if all the z initial partial solutions have gaps equal to zero, then an optimal solution is obtained by using, iteratively, the *ordered combination* operator.

The procedure *OFS*, which finds an *ordered z-family of solutions*, first orders the jobs so that $p_1 \geq p_2 \geq \ldots \geq p_n$ and computes z as the greatest index of the jobs so that $\sum\limits_{i=1,\dots,z\leq n} p_i \leq \max\{p_1, p_m + p_{m+1}, \frac{1}{m}\sum\limits_{i=1,\dots,n} p_i\}$. Then, by using as seeds the first z jobs, the procedure initializes z partial solutions. Finally, *OFS* processes the remaining jobs, by assigning each job i iteratively to the partial solution which corresponds to the greatest Δ^r values, e.g. Δ^k, if $\Delta^k \geq p_i$. Otherwise *OFS* uses the job i as seed to initialize a new partial solution.

The procedure can be formally described as follows.

Procedure *OFS*

Initialisation

- Order the jobs so that $p_1 \geq \ldots \geq p_i \geq \ldots \geq p_n$. Set z the greatest index of the jobs so that $\sum\limits_{i=1,\dots,z\leq n} p_i \leq \max\{p_1, p_m + p_{m+1}, \frac{1}{m}\sum\limits_{i=1,\dots,n} p_i\}$;
- Set $\mathfrak{I}^1$=$\{I_1^1 = \{1\}, I_2^1 = \emptyset, \dots, I_j^1 = \emptyset, \dots, I_m^1 = \emptyset\}$, $\mathfrak{I}^2$=$\{I_1^2 = \{2\}, I_2^2 = \emptyset, \dots, I_j^2 = \emptyset, \dots, I_m^2 = \emptyset\}$,..., $\mathfrak{I}^z$=$\{I_1^z = \{z\}, I_2^z = \emptyset, \dots, I_j^z = \emptyset, \dots, I_m^z = \emptyset\}$, and $G^1 = \{p_1^1 = p_1, p_2^1 = 0, \dots, p_j^1 = 0, \dots, p_m^1 = 0\}$, $G^2 = \{p_1^2 = p_2, p_2^2 = 0, \dots, p_j^2 = 0, \dots, p_m^2 = 0\}$,..., $G^z = \{p_1^z = p_z, p_2^z = 0, \dots, p_j^z = 0, \dots, p_m^z = 0\}$;
- Set $\Delta^r = p_r$, $r = 1, ..., z$;
- Set $\mathfrak{P} = \{\mathfrak{I}^1, \dots, \mathfrak{I}^r, \dots, \mathfrak{I}^z\}$ ($\mathfrak{P}$ is ordered so that $\Delta^1 \geq \ldots \geq \Delta^r \geq \ldots \geq \Delta^z$).

Construction.

For $i = z+1, \dots, n$

a) select $\mathfrak{I}^1$ (since $\Delta^1 = \max\limits_{r=1,\dots,z} \Delta^r$);

b) If $\Delta^1 \geq p_i$ then
-set $I_m^1 = I_m^1 \cup \{i\}$ and $p_m^1 = p_m^1 + p_i$;
-sort the elements of G^1 so that $p_1^1 \geq \ldots \geq p_j^1 \geq \ldots \geq p_m^1$;
-arrange $\mathfrak{I}^1$ so that p_j^1 is the total time required by the jobs belonging to I_j^1, for $j = 1, \ldots, m$;
-set $\Delta^1 = p_1^1 - p_m^1$ and sort $\mathfrak{P}$ so that $\Delta^1 \geq \ldots \geq \Delta^r \geq \ldots \geq \Delta^z$;
Otherwise
-set $z = z + 1$, $\mathfrak{I}^z = \{I_1^z = \{i\}, I_2^z = \emptyset, \ldots, I_j^z = \emptyset, \ldots, I_m^z = \emptyset\}$, $\mathfrak{P} = \mathfrak{P} \cup \mathfrak{I}^z$;
-set $G^z = \{p_1^z = p_i, p_2^z = 0, \ldots, p_j^z = 0, \ldots, p_m^z = 0\}$;
-set $\Delta^z = p_i$, and sort $\mathfrak{P}$ so that $\Delta^1 \geq \ldots \geq \Delta^r \geq \ldots \geq \Delta^z$;
End If $\Delta^1 \geq p_i$.
End For i.

The *OFS* procedure finds an *ordered z-family of solutions* $\mathfrak{P} = \{\mathfrak{I}^1, \ldots, \mathfrak{I}^r, \ldots, \mathfrak{I}^z\}$ such that

1) p_1^r is a singleton, $r = 1, \ldots, z$
2) $\max\limits_{r=1,\ldots,z} \Delta^r = \Delta^1 \leq \min\limits_{r=1,\ldots,z} p_1^r$.

It is easy to show that the algorithm *PSC* runs in $O(nlog(n))$-time which is the running time of the procedure *OFS*.

4. Computational investigation

The *PSC* algorithm was tested on three different families of instances and compared with the classical *LPT* heuristic of Graham, the improvement *3-PHASE* heuristic of França et al. and the improvement *1-SPT* algorithm of Frangioni et al.

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

$$L_2 = \max\left\{ \left\lceil \frac{1}{m} \sum_{i=1,\ldots,n} p_i \right\rceil, p_1, p_m + p_{m+1} \right\},$$

where $p_1 \geq p_2 \geq \ldots \geq p_n$.

In Tables 1-3, columns *PSC*, *LPT*, *3-PHASE* and *1-SPT* describe the results of *PSC* algorithm, of the *LPT* heuristic, of the algorithm of França et al., and of the algorithm designed by Frangioni et al., respectively. Likewise the number of instances in which the algorithms obtain the makespan equal to the lower bound, representing instances solved to optimality, was reported in column o.

The results reported by Frangioni et al. (1999) for the *3-PHASE* and *1-SPT* algorithms were utilized in this paper, as the same instances were used here. These results does not include the column o.

The Instances

Three different families of instances were taken from the literature.

In the first two families the number of machines m are $5, 10, 25$, the number of jobs n are $50, 100, 500, 1000$ (for $m = 5$ and $n = 10$ was also tested), and the interval for the integer processing times were $[1, 100], [1, 1000]$, and $[1, 10000]$.

Ten instances were 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 were generated by using an uniform distribution.

The generator of the second family (NON-UNIFORM), which was presented by Frangioni et al. (1999 and 2004), when an interval $[a, b]$ of the processing times is given, 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 at the URL

http://www.di.unipi.it/di/groups/optimize/Data/index.html

The last family of instances were derived from several difficult bin packing instances, which are available at the OR-Library of J.E. Beasley, at the URL

http ://mscmga.ms.ic.ac.uk/jeb/orlib/binpackinfo.html

In this family, denoted BINPACK, 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.

Computational results

The results for UNIFORM instances are shown in Table 1 for the three subsets of instances with processing times in $[1, 100], [1, 1000]$, and $[1, 10000]$. UNIFORM instances are known to be efficiently approached with *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 *PSC* algorithm offers significant improvement over *LPT*; in 15 out of 30 cases, the average relative error of *PSC* is comparable with respect to the more accurate *3-PHASE*.

The results for NON-UNIFORM instances are shown in Table 2 for the three subsets of instances with processing times in $[1, 100]$, $[1, 1000]$, and $[1, 10000]$. These instances are, in all the cases, more difficult than the UNIFORM instances, as greater gaps remain in all three algorithms examined. The *PSC* algorithm consistently outperformed *LPT* algorithm, generating much smaller gaps. In addition, compared with the more accurate *3-PHASE* algorithm, *PSC* always obtained comparable gaps.

The results in Table 3 show that the *PSC* algorithm can be successfully applied to BINPACK instances as the solutions obtained were better than these provided by the *LPT* algorithm and comparable with these given by *1-SPT*.

		$\mathbf{p_j \in [1,100]}$					$\mathbf{p_j \in [1,1000]}$					$\mathbf{p_j \in [1,1000]}$				
		LPT		**3-PHASE**	**PSC**		**LPT**		**3-PHASE**	**PSC**		**LPT**		**3-PHASE**	**PSC**	
m	**n**	**gap**	**o.**	**gap**	**gap**	**o.**	**gap**	**o.**	**gap**	**gap**	**o.**	**gap**	**o.**	**gap**	**gap**	**o.**
5	10	3.76e-02	3	3.26e-02	3.76e-02	3	4.04e-02	2	4.03e-02	4.39e-02	2	6.78e-03	8	6.78e-03	6.78e-03	8
	50	4.23e-03		1.96e-04	1,21e-03	5	4.27e-03		2.48e-04	1,68e-03		5.44e-03		5.44e-04	1,97e-03	
	100	1.46e-03	3	0.00e-00	0.00e-00	10	1.22e-03		6.03e-05	2.89e-04	1	9.84e-04		7.78e-05	4.10e-04	
	500	0.00e-00	10	0.00e-00	0.00e-00	10	2.05e-05	4	0.00e-00	4.19e-06	8	4.49e-05		3.96e-07	2.37e-06	4
	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	3.02e-07	7
10	50	1.60e-02	1	2.35e-03	8.35e-03	2	2.80e-02		4.43e-03	1.67e-02		2.07e-02		4.81e-03	1.59e-02	
	100	4.93e-03	2	0.00e-00	1.60e-03	5	4.76e-03		3.43e-04	3.01e-03		5.63e-03		2.48e-04	2.54e-03	
	500	4.10e-05	9	0.00e-00	0.00e-00	10	1.61e-04		0.00e-00	6.82e-05	1	1.59e-04		2.38e-06	6.41e-05	
	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	6.84e-06	
25	50	3.35e-02	2	3.42e-02	3.35e-02	2	2.46e-02	5	2.33e-02	2.46e-02	5	2.72e-02	2	2.08e-02	2.72e-02	2
	100	2.52e-02		3.62e-03	2.40e-02		2.23e-02		4.04e-03	3.01e-02		3.26e-02		4.33e-03	2.25e-02	
	500	5.04e-04	5	0.00e-00	3.06e-04	7	1.06e-03		0.00e-00	4.63e-04		1.31e-03		2.28e-05	6.60e-04	
	1000	1.01e-04	8	0.00e-00	0.00e-00	10	2.81e-04		0.00e-00	1.96e-04		3.55e-04		3.52e-06	1.04e-04	

Table 1 - Computational results for UNIFORM instances

		$\mathbf{p_j \in [1,100]}$				$\mathbf{p_j \in [1,1000]}$				$\mathbf{p_j \in [1,1000]}$			
		LPT		**3-PHASE**	**PSC**	**LPT**		**3-PHASE**	**PSC**	**LPT**		**3-PHASE**	**PSC**
m	**n**	**gap**	**o.**	**gap**	**gap**	**gap**	**o.**	**gap**	**gap**	**gap**	**o.**	**gap**	**gap**	**o.**
5	10	5.21e-03	2	7.52e-03	5.21e-03	2	6.13e-03	1	6.13e-03	6.13e-03	1	6.17e-03	1	6.18e-03	6.17e-03	1
	50	1.69e-02		1.54e-02	7.50e-03		1.75e-02		1.59e-02	8.70e-03		1.76e-02		1.58e-02	8.85e-03	
	100	1.67e-02		6.05e-03	6.75e-03		1.70e-02		6.52e-03	7.71e-03		1.70e-02		6.64e-03	7.81e-03	
	500	5.86e-04		2.03e-03	0.00e-00	10	6.35e-04		0.00e-00	6.39e-06	6	6.42e-04		0.00e-00	3.81e-05	1
	1000	1.44e-04		1.25e-03	0.00e-00	10	1.78e-04		1.88e-04	0.00e-00	10	1.78e-04		0.00e-00	8.61e-06	5
10	50	1.76e-02		1.42e-02	1.44e-02		1.82e-02		1.42e-02	1.49e-02		1.82e-02		1.42e-02	1.50e-02	
	100	1.72e-02		7.31e-03	1.21e-02		1.77e-02		7.18e-03	1.30e-02		1.78e-02		7.25e-03	1.31e-02	
	500	1.00e-02		6.63e-03	4.43e-03		1.01e-02		6.46e-03	4.80e-03		1.02e-02		6.65e-03	4.87e-03	
	1000	3.83e-04	1	2.74e-03	0.00e-00	10	4.12e-04		4.53e-04	9.57e-06	6	4.10e-04		0.00e-00	1.01e-05	1
25	50	5.30e-03	4	5.28e-03	5.30e-03	4	4.03e-03	4	4.03e-03	4.03e-03	4	4.09e-03	3	4.09e-03	4.09e-03	3
	100	1.77e-02		7.44e-03	1.55e-02		1.81e-02		7.50e-03	1.71e-02		1.81e-02		7.58e-03	1.72e-02	
	500	9.79e-03		6.43e-03	7.61e-03		1.00e-02		6.59e-03	7.78e-03		1.01e-02		6.58e-03	7.82e-03	
	1000	9.30e-03		6.16e-03	6.99e-03		9.38e-03		6.18e-03	7.00e-03		9.40e-03		6.20e-03	7.04e-03	

Table 2 - Computational results for NON-UNIFORM instances

		LPT	***1-SPT***	***PSC***
m	**n**	**gap**	**gap**	**gap**
48	120	1.16e-01	2.30e-02	3.09e-02
50	120	1.33e-01	1.90e-02	2.84e-02
102	250	1.20e-01	2.28e-02	2.74e-02
102	250	1.42e-01	1.94e-02	2.14e-02
203	500	1.26e-01	2.48e-02	3.07e-02
200	500	1.41e-01	1.33e-02	2.53e-02
402	1000	1.32e-01	1.80e-02	3.07e-02
399	1000	1.35e-01	2.67e-02	2.35e-02

Table 3 - Computational results for BINPACK instances

5. Conclusions

In this paper an $n \log n$ algorithm for solving parallel machine scheduling problem to minimize the makespan is proposed. The algorithm presented is capable of producing good quality solutions for all classes of instances used in the computational investigation. Moreover, this algorithm produces less average relative errors than the *LPT* algorithm, and generally the average relative errors are comparable with those generated by the *3-PHASE* and 1-SPT improvement heuristics.

6. Acknowledgments

We thank F.M. Muller and E. Necciari for having provided the generators and the instances used for the computational testing.

References

[1] E.J. Anderson, C.A. Glass and C.N. Potts, *Machine scheduling*, in Local Search in Combinatorial Optimization, Aarts and Lenstra eds.,Wiley, New York, (1997), 361-414.

[2] B. Chen, C.N. Potts and G.J. Woeginger, *A review of machine scheduling: Complexity, algorithms and approximability*, in Handbook of Combinatorial Optimization, Du D.Z. and Pardalos P. eds., Vol. 3, Kluwer Academic Publishers, Dordrecht, (1998), 21-169.

[3] T. Cheng and C. Sin, *A state of the art review of parallel machine scheduling research*, European Journal of Operation Research, 47(1990), 271-292.

[4] E.G. Jr. Coffman, M.R. Garey and D.S. Johnson, *An application of bin-packing to multiprocessor scheduling*, SIAM Journal of Computing, 7(1978), 1-17.

[5] P.M. França, M. Gendrau, G. Laporte and F.M. Muller, *A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective*, Computers Ops.Res., 21(2)(1994), 205-210.

[6] A. Frangioni, M.G. Scutellà and E. Necciari, *Multi-exchange algorithms for the minimum makespan machine* , Technical Report:99-22, Dipartimento di Matematica, University of Pisa, Italy(december 1999).

[7] A. Frangioni, E. Necciari and M.G. Scutellà, *A multi-exchange neighborhood for minimum makespan machine scheduling problems*, Journal of Combinatorial optimization, 8(2004), 195-220.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA (1979).

[9] R.L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal, 45(1966), 1563-1581.

[10] R.L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM Journal of Applied Mathematics, 17(1969), 416-429.

[11] 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(1979), 287-326.

[12] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, *Sequencing and scheduling: algorithms and complexity in logistics of production and inventory*, in Handbooks in Operation Research and Management Science, Vol. 4, Elsevier Science Publishers, (1993), 445-522.

[13] G. Paletta and P. Pietramala, *A new approximation algorithm for the non-preemptive scheduling of independent jobs on identical parallel processors*, SIAM J. Discrete Math., 21 (2007), 313-328.

[14] J.D. Ullman, *Complexity of sequencing problems*, in Computer and Job Shop Scheduling Theory, E.G. Coffman ed., Wiley, New York, (1976), 139-164.

Received: June 20, 2007