Computers & Operations Research 39 (2012) 736–745

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Two-stage hybrid flow shop with precedence constraints and parallel machines at second stage

Sergiu Carpov [a,b,*], Jacques Carlier [b], Dritan Nace [b], Renaud Sirdey [a]

[a] CEA, LIST, Embedded Real Time Systems Laboratory, Point Courrier 94, 91191 Gif-sur-Yvette Cedex, France
[b] UMR CNRS 6599 Heudiasyc, Université de Technologie de Compiègne, Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France

ARTICLE INFO

Available online 2 June 2011

Keywords:
Hybrid flow shop
Precedence relations
Randomized list scheduling
Multiprocessor scheduling

ABSTRACT

This study deals with the two-stage hybrid flow shop (HFS) problem with precedence constraints. Two versions are examined, the classical HFS where idle time between the operations of the same job is allowed and the no-wait HFS where idle time is not permitted. For solving these problems an adaptive randomized list scheduling heuristic is proposed. Two global bounds are also introduced so as to conservatively estimate the distance to optimality of the proposed heuristic. The evaluation is done on a set of randomly generated instances. The heuristic solutions for the classical HFS in average are *provably* situated below 2% from the optimal ones, and on the other hand, in the case of the no-wait HFS the average deviation is below 5%.

1. Introduction

This work considers the hybrid flow shop problem under precedence constraints. More precisely the two-stage hybrid flow shop $HF(1,P_m)$ with precedence constraints at the second stage is studied, by abuse of notation we denote it HFS in what follows. Assume a set of n jobs has to be processed in two stages. There is only one machine for the first stage and m identical parallel machines for the second stage. Each job $i \in \{1,\ldots,n\}$ consists of two operations: the first operation of duration $a_i > 0$ is executed at the first stage, and afterwards the second operation of duration $b_i > 0$ is executed at the second stage. No preemption is allowed in operation execution. The precedence constraints of the operations at the second stage are given by a directed acyclic graph $G = (V,E)$, where V represents the set of jobs and E gives the dependence relations between those jobs. There are no precedence constraints between the operations at the first stage.

The objective is to minimize the *maximum completion time* or *makespan*. Two different cases of HFS can be distinguished: the *no-wait* HFS when once a job has started it is executed on all the stages without being interrupted (the end time of the first stage operation coincides with the start time of the second stage operation) and the *classical* HFS when no such constraint is imposed. In the $\alpha|\beta|\gamma$ notation the flow shop problems we examine are $HF(1,P_m)|G_1 = \emptyset, G_2 = G|C_{max}$ and $HF(1,P_m)|G_1 = \emptyset, G_2 = G, \text{no} - \text{wait}|C_{max}$.

Despite that no precedence relations are defined for the first stage operations, the second stage constraints can be extended over the first stage because they are dominating the order in which the first stage operations are executed. This fact is obvious in the case of no-wait HFS. On the other hand it can be easily shown that for any given solution in a classical HFS, rescheduling the first stage operations following the same second stage schedule does not change the solution value. Hence, in what follows we consider that if a second stage operation must be executed after another second stage operation then the corresponding first stage operations must follow the same order.

A practical application of the HFS problem arises in modeling the execution of an algorithm on a parallel computer. Each algorithm task can be viewed as two consecutive operations, the first one is the loading of the data used by the task from the external memory and the second one is the task execution itself. Usually in a parallel computer the memory accesses are done sequentially, so only one data loading can be done at a time, whereas the execution of the tasks can be done concurrently on the available processors. Hence the data loading corresponds to the first stage operation in the HFS problem, and the task execution corresponds to the second stage operation. Second stage precedence relations between the operations are equivalent to the partial order of algorithm tasks and reflect the internal data dependencies (amongst other dependencies). In order to limit the data buffering, the execution of a task has to start when its data loading is finished, this corresponds to the no-wait case of the

* Corresponding author at: CEA, LIST, Embedded Real Time Systems Laboratory, Point Courrier 94, 91191 Gif-sur-Yvette Cedex, France. Tel.: +33 1 69 08 60 48.
E-mail addresses: sergiu.carpov@cea.fr (S. Carpov), carlier@hds.utc.fr (J. Carlier), dnace@hds.utc.fr (D. Nace), renaud.sirdey@cea.fr (R. Sirdey).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.05.020

HFS, whereas the classical HFS corresponds to the case when no space limit is imposed on the data buffering.

The paper is organized as follows: after a brief description of related works in Section 2, two global lower bounds are introduced in Section 3. Section 4 presents a list scheduling heuristic, and, in Section 5 we describe a randomized version of this algorithm. In Section 6 the lower bounds and heuristics performances are compared using randomly generated instances and Section 7 concludes the paper.

2. Related works

The literature on the hybrid flow shop problem under precedence constraints is quite scarce, even though a lot of work exist on the hybrid flow shop and on the flow shop with precedence relations. For a review of the plentiful work on the hybrid flow shop problem we refer to [1,2]. We shall note that most of the work is done for the general m-stage hybrid flow shop, nevertheless many authors tried to adapt the Johnson algorithm for the two-stage flow shop. A model close to ours, the two-stage hybrid flow shop with parallel machines at first stage only is studied in [3]. The authors determine the optimal ordering at the second stage given a scheduling of jobs on first stage and introduce some interesting lower bound concepts.

Although less represented in the literature, the flow shop problem under precedence constraints is quite well studied. In [4] the authors provide a classification of two and three machine flow shop problems under machine-dependent precedence constraints. Different models of shop scheduling problems with precedence constraints are considered in [5]. In their study the authors introduce two types of precedence constraints and provide complexity results and some polynomial time algorithms for shop scheduling models. The authors of [6] propose to reduce the job shop problem to a flow shop problem under precedence constraints, and introduce several modified flow shop heuristics for solving the flow shop problem constrained by precedence relations.

The hybrid flow shop problem under precedence constraints is studied in a few papers [7–9,1], from an applicative point of view. In the studies mentioned above some heuristics are proposed. The authors are using stage-independent precedence relations between the jobs and different optimization criteria.

3. Lower bounds

Without loss of generality we suppose, in what follows, that the digraph $G=(V,E)$ describing the precedence relations between the operations at the second stage contains one source vertex, denoted 0, and one sink vertex, denoted $*$, with zero processing times. Also we suppose that the number of jobs is greater than the number of available second stage machines, $n>m$.

3.1. Global lower bound 1

Some concepts of the following lower bound were introduced in [10] for the hybrid flow shop problem. We have adapted it in order to take advantage of the second stage precedence relations.

$$GLB1=\max(GLB1^1,GLB1^2)$$

In the first part $GLB1^1$ of the bound we take into account that there is inevitably an idle time at the second stage machines during the execution of the first $m+1$ jobs. During this idle time the first stage operations of the respective jobs are executed (see Fig. 1 for an illustration).

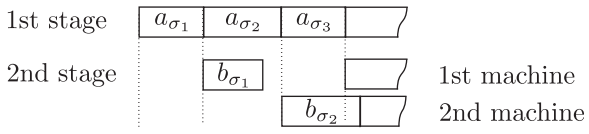

Fig. 1. Second stage idle time needed to execute first stage operations (in this example the total idle time equals to $(a_{\sigma_1}+a_{\sigma_2}+a_{\sigma_3}-b_{\sigma_1})+(a_{\sigma_1}+a_{\sigma_2})$).

Let $\sigma_1,\ldots,\sigma_{m+1}$ be the ordering of the first executed $m+1$ jobs at the first stage, σ_i represents the job in position i. For any precedence constraint between two jobs i,j, thus any edge $(i,j)\in E$ of graph G, if both jobs i,j belong to the ordering then relation $\sigma_i^{-1}<\sigma_j^{-1}$ must be satisfied (σ_i^{-1} is the position of job i). The precedence relations can be rephrased as: operation σ_1 has to be a successor of the source node 0 such that σ_1 has only one predecessor (which is the source node itself), operation σ_k must satisfy $\mathrm{pred}(\sigma_k)\subseteq\{0,\sigma_1,\ldots,\sigma_{k-1}\}$, and so on. Here $\mathrm{succ}(i_1,\ldots,i_k)$, $\mathrm{pred}(i_1,\ldots,i_k)$, represents the union of successors, respectively, predecessors, of vertices $i_1,\ldots,i_k$ in the graph G.

The idle time at the second stage machine where job σ_k is executed is at least $\sum_{i=1}^{k}a_{\sigma_i}+\max(\sum_{i=k+1}^{m+1}a_{\sigma_i}-b_{\sigma_k},0)$. For the ordering $\sigma_1,\ldots,\sigma_{m+1}$ the total second stage idle time is

$$Z_1=\sum_{k=1}^{m}\left(\sum_{i=1}^{k}a_{\sigma_i}+\max\left(\sum_{i=k+1}^{m+1}a_{\sigma_i}-b_{\sigma_k},0\right)\right)$$

The sum between the minimum possible idle time Z_1 and the total amount of the second stage jobs duration divided by the number of available second stage machines gives a lower bound on the execution time. As all processing times are integers the lower bound should have also an integer value, a ceiling operator $\lceil\ \rceil$ is used for this purpose:

$$GLB1^1=\left\lceil\frac{1}{m}\left(Z_1+\sum_{i=1}^{n}b_i\right)\right\rceil$$

In order to find the sequence $\sigma_1,\ldots,\sigma_{m+1}$ which satisfies the precedence constraints and minimizes Z_1, the following combinatorial problem must be solved:

$$Z_1=\text{Minimize}\quad\sum_{k=1}^{m}\left(\sum_{i=1}^{k}a_{\sigma_i}+\max\left(\sum_{i=k+1}^{m+1}a_{\sigma_i}-b_{\sigma_k},0\right)\right)$$
$$\text{s.t.}\quad\mathrm{pred}(\sigma_k)\subseteq\{0,\sigma_1,\ldots,\sigma_{k-1}\}$$

The following relaxation makes this problem solvable in polynomial time (here relation $\mathrm{anc}(l)$ gives the ancestor vertices of vertex l):

$$Z_1'=\underset{\sigma_k^1}{\text{Minimize}}\sum_{k=1}^{m}a_{\sigma_k^1}(m-k+1)+\underset{\sigma_k^2}{\text{Minimize}}\sum_{k=1}^{m}\max\left(\sum_{i=k+1}^{m+1}a_{\sigma_k^1}-b_{\sigma_k^2},0\right)$$
$$\text{s.t. }|\mathrm{anc}(\sigma_k^l)|\le k,\ l=1,2$$

The relaxation consists in minimizing the two parts of the objective function separately. First, an ordering σ^1 that minimizes the left hand side of Z_1' and afterwards a new ordering σ^2 which minimizes the right hand side of objective, should be found. The solution of the relaxed problem can be used for lower bound calculation in place of the initial problem solution because $Z_1'\le Z_1$. Algorithm 1 finds the solution Z_1' of the relaxed problem. We shall note that in our experiments, we have obtained a deviation between the optimal global lower bound (calculated using Z_1) and the relaxed version (calculated using Z_1') less than 0.2%. This fact indicates that there is no much benefit from using

the optimal calculation for Z_1 when compared to relaxed computation Z_1', especially that in the majority of cases ($>75\%$) the same solution is found.

Algorithm 1. Algorithm for finding the optimal solution of the relaxed problem used in $GLB1^1$ calculation.

$B^1, B^2 = \emptyset$
for $k=1$ **to** $m+1$ **do**
  $\sigma_k^1 = \text{argmin} a_i$, such that $|\text{anc}(i)| \le k$ and $i \notin B^1$
  $B^1 = B^1 \cup \{\sigma_k^1\}$
  $\sigma_k^2 = \text{argmax} b_i$, such that $|\text{anc}(i)| \le k$ and $i \notin B^2$
  $B^2 = B^2 \cup \{\sigma_k^2\}$
end for
$Z_1' = 0$
$S = a_{\sigma_{m+1}^1}$
for $k=m$ **to** 1 **do**
  $Z_1' = Z_1' + a_{\sigma_k^1} \cdot (m-k+1) + \max(S - b_{\sigma_k^2}, 0)$
  $S = S + a_{\sigma_k^1}$
end for

The second part $GLB1^2$ of the bound takes into consideration the fact that the execution cannot finish before all the operations at the first stage are processed. Additionally, in the best case, the last operations executed at the second stage are those that are predecessors of the sink node and have minimal processing times. Refer to Fig. 2 for an illustration of such configuration.

Let $\sigma_1, \ldots, \sigma_m$ be the last m jobs executed at the second stage in reverse order, that is σ_1 is the last job, σ_2 is the penultimate one, etc. Like in the previous case, job precedence relations must be satisfied. Thus, the job in position k, $k=1,\ldots,m$, must satisfy $\text{succ}(\sigma_k) \subseteq \{*, \sigma_1, \ldots, \sigma_{k-1}\}$.

Job σ_k can start at the second stage, only after all the first stage operations which are executed before are finished. In this case, the completion time of job σ_k is at least $\sum_i a_i + Z_2^k$, where $Z_2^k = b_{\sigma_k} - \sum_{i=1}^{k-1} a_i$ represents the exceedance of job k over the total first stage workload.

A lower bound for the HFS problem is given by (1), where Z_2 represents the least possible exceedance for any sequence of final jobs. In order to find the ordering of last m operations for which Z_2 is minimal the combinatorial problem (2) must be solved.

$$GLB1^2 = \sum_{i=1}^{n} a_i + Z_2 \tag{1}$$

$$Z_2 = \text{Minimize} \quad \max_{k=1,\ldots,m} \left(b_{\sigma_k} - \sum_{i=1}^{k-1} a_{\sigma_i} \right)$$
$$\text{s.t.} \quad \text{succ}(\sigma_k) \subseteq \{*, \sigma_1, \ldots, \sigma_{k-1}\} \tag{2}$$

Proposition 1. *Optimal solution of optimization problem* (2) *is given by the recurrent relation*

$$\sigma_k = \arg \min_{\substack{i \notin \{*, \sigma_1, \ldots, \sigma_{k-1}\} \\ \text{succ}(i) \subseteq \{*, \sigma_1, \ldots, \sigma_{k-1}\}}} b_i$$

for all $k = 1, \ldots, m$.

Proof. Suppose that $\sigma_1, \ldots, \sigma_m$ is the optimal solution of the problem having value Z_2, and also, suppose that there exists an operation p, $\text{succ}(p) = *$, such that $b_p < b_{\sigma_1}$:

1. If $p \notin \{\sigma_1, \ldots, \sigma_m\}$ a new solution $p, \sigma_1, \ldots, \sigma_{m-1}$ (see Fig. 3 for an illustration) will have the following value:
$$Z_2' = \max\left(b_p, b_{\sigma_1} - a_p, \ldots, b_{\sigma_{m-1}} - \sum_{i=1}^{m-2} a_{\sigma_i} - a_p \right)$$
$$= \max\left(b_p, \max_{k=1}^{m-1} \left(b_{\sigma_k} - \sum_{i=1}^{k-1} a_{\sigma_i} \right) - a_p \right) \le Z_2$$
The last result, $Z_2' \le Z_2$, contradicts the fact that Z_2 is the optimal solution.
2. If $p \in \{\sigma_1, \ldots, \sigma_m\}$ a new solution can be obtained by moving operation p before σ_1 (thus, p will be the last executed job). In an analogous way, we prove that the new solution is better.

We deduce that in an optimal solution $\sigma_1 = \text{argmin}_{\forall i, \text{succ}(i) = *} b_i$. Applying the same methodology the proposition is proved by induction. □

Algorithm 2 finds the optimal solution for the minimization problem in polynomial time using the previous result.

Algorithm 2. Algorithm for finding the optimal solution Z_2 of the problem used in $GLB1^2$ calculation.

$A = \{i | i \in \text{pred}(*), \text{succ}(i) = *\}$
$B = \emptyset$
for $k=1$ **to** m **do**
  $\sigma_k = \text{argmin} b_i$, such that $i \in A$ and $i \notin B$
  $B = B \cup \{\sigma_i\}$
  $A = A \cup \{i | i \in \text{pred}(\sigma_k), \text{succ}(i) \subseteq \{*, \sigma_1, \ldots, \sigma_{k-1}\}\}$
end for

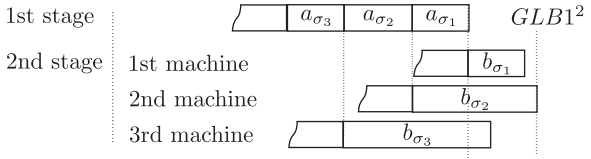

Fig. 2. Final moments of a HFS with three machines at the second stage.

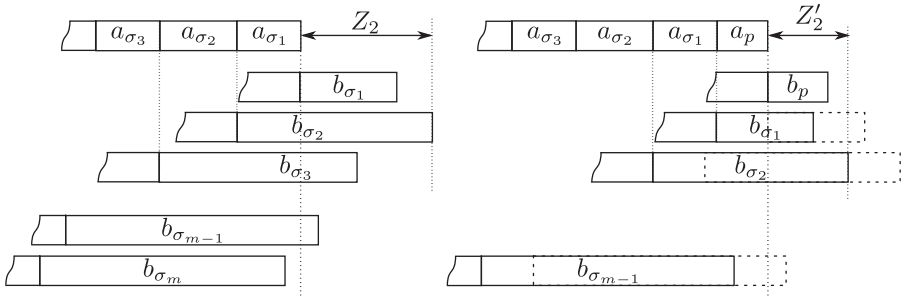

Fig. 3. Solution Z_2' compared to the initial one Z_2. On the righthand side in dashed line are presented execution intervals of operations in the initial solution.

```
8:    Z2=0
9:    S=0
10:   for k=1 to m do
11:     Z2 = max(Z2,b_σk−S)
12:     S=S+a_k
13:   end for
```

3.2. Global lower bound 2

In this subsection we introduce a global lower bound based on release times (heads) and delivery times (tails) adjustments. Let us assume that operation i cannot start earlier than its release date r_i, it is processed for either a_i or b_i time in function of the stage and must remain in the system for at least q_i time, which is the tail of operation i. In order to differentiate the first stage heads and tails from the second stage ones they are superscripted, so r_i^I, q_i^I are the heads and tails at the first stage and, respectively, r_i^{II}, q_i^{II} at the second stage. We use heads and tails instead of release dates and deadlines because many constraint concepts can be symmetrically expressed for heads and tails.

A straightforward lower bound is (3), the first stage release dates and tails are not taken into account because they are dominated by the second stage heads and tails:

$$GLB2 = \max_i(r_i^{II} + b_i + q_i^{II}) \tag{3}$$

In what follows we introduce several constraints that the heads and tails must satisfy. Using constraint propagation techniques the heads and tails are iteratively adjusted until no modification is observed, the obtained *GLB2* is a lower bound to the HFS problem.

3.2.1. Inter-stage precedence relations

In a two-stage flow shop the first stage operation of a job i must finish before its second stage operation starts: $r_i^{II} \geq r_i^I + a_i$. In the case of a no-wait flow shop this relation is more constrained by the fact that no idle time is permitted between the stages, so for the no-wait HFS we have $r_i^{II} = r_i^I + a_i$.

The same type of relations can be deduced for job tails: $q_i^I \geq q_i^{II} + b_j$ for the classic HFS and $q_i^I = q_i^{II} + b_j$ for the no-wait case.

3.2.2. Jobs precedence relations

The precedence relation graph $G=(V,E)$ for second stage operation is translated into the following constraint: $r_i^{II} \geq r_j^{II} + b_j$ for all $j \in \text{pred}(i)$. Symmetrically for job tails: $q_i^{II} \geq q_j^{II} + b_j$ for all $j \in \text{succ}(i)$. For the no-wait HFS the second stage precedence relations directly influence the partial order of the first stage operations because of relations introduced in the previous section. In the case of classic HFS things are a little bit different, but it can be easily proved that the second stage precedence relations are dominating over the first stage ones.

3.2.3. Cumulative previous work

As said above, the second stage precedence constraints define a partial ordering over the first stage operations, thus before the execution of the first stage operation i can start, all its first stage ancestors, defined by the second stage precedence constraints, must be completed. The release date r_i^I must be larger than the minimum makespan of a one-machine scheduling problem with release dates composed of ancestors of operation i. Let $C_{max}^{r^I}(i)$ be the optimal makespan of problem $1\ |r_j|C_{max}$ for operations $j \in \text{anc}(i)$ with release dates r_j and processing times a_j, then the head of the first stage operation i must satisfy $r_i^I \geq C_{max}^{r^I}(i)$.

The one-machine scheduling with release dates for jobs $j_1, \ldots, j_p$ is solved in polynomial time using the recurrent relation (Jackson's rule) $c_{j_k} = \max(r_{j_k}, c_{j_{k-1}}) + a_{j_k}$ with initial conditions $c_{j_1} = r_{j_1} + a_{j_1}$ and $r_{j_1} \leq r_{j_2} \leq \cdots \leq r_{j_p}$. Completion time c_{j_p} of the last job is the solution of the problem.

A straightforward relaxation of this constraint is $r_i^I \geq \sum_{j \in \text{anc}(i)} a_j$ which has a linear time computation, but it produces weaker release date bounds.

A constraint for the tails of the first stage operation is obtained in a similar way. The tail of operation i must satisfy $q_i^I \geq C_{max}^{q^I}(i)$ where $C_{max}^{q^I}(i)$ is the solution of the one-machine scheduling problem for descendants $j \in \text{desc}(i)$ with release date q_j and processing times a_j. A direct relaxation is obtained equivalently $q_i^I \geq \sum_{j \in \text{desc}(i)} a_j + \min_{j \in \text{desc}(i)} q_j^I$. In the above expression, the "min" term is added because the tails are not necessary zero as in the case with release dates.

In order to deduce equivalent relations for the heads and tails of the second stage operations, the parallel processor scheduling problem $Pm|r_i|C_{max}$ should be solved. The later problem is $\mathcal{NP}$-hard [11], thus a polynomial algorithm for solving it does not exist (unless $\mathcal{P}=\mathcal{NP}$). The parallel processor scheduling problem can be relaxed to a one-machine scheduling problem by dividing the processing times of the jobs by the number of processors. That is to say, we consider that a job can be executed simultaneously on all of the available processors.

For the second stage operation i, we consider the one-machine scheduling problem $1\ |r_j|C_{max}$ for ancestor jobs of i with processing times $b'_j = b_j/m$ and release dates r_j^{II} for any $j \in \text{anc}(i)$. Let $C_{max}^{r^{II}}(i)$ be the optimal makespan of the above problem. The release date of the second stage operation i must satisfy $r_i^{II} \geq \lceil C_{max}^{r^{II}}(i) \rceil$, a ceiling operator is used because the release date must be integer. A linear relaxation of the above constraint is

$$r_i^{II} \geq \left\lceil \frac{\sum_{j \in \text{anc}(i)} b_j}{m} \right\rceil + \min_{j \in \text{anc}(i)} r_j^{II}$$

Symmetrically for the tails of the second stage operation i the following constraint is deduced $q_i^{II} \geq C_{max}^{q^{II}}(i)$. Where $C_{max}^{q^{II}}(i)$ is the optimal solution of the one-machine scheduling problem for the descendants $j \in \text{desc}(i)$ of operation i with processing times $b'_j = b_j/m$ and release dates q_j. Equivalently the following linear relaxation is inferred, here we have $\min_{j \in \text{desc}(i)} q_i^{II} = q_*^{II} = 0$:

$$q_i^{II} \geq \left\lceil \frac{\sum_{j \in \text{desc}(i)} b_j}{m} \right\rceil$$

3.2.4. Jackson's preemptive schedule

Jackson's preemptive schedule (JPS) was introduced in [12]. It gives the optimal makespan for the preemptive one-machine scheduling with release dates and delivery times $1\ |r_i, q_i, pmnt|C_{max}$. The obtained makespan value is a tight lower bound for the non-preemptive problem $1\ |r_i, q_i|C_{max}$. JPS is the list schedule found by prioritizing the jobs with the most remaining work. The jobs are examined in increasing order of their release dates. At time instant t the job with the largest delivery time among the available jobs is scheduled, even if another job is in execution.

In *GLB2* calculation the HFS problem is relaxed to $1\ |r_i^I, q_i^I|C_{max}$ by dropping out the second stage and looking only at the first stage problem. The JPS is then used to adjust the global lower bound *GLB2*.

The JPS can also be used to adjust the heads and tails of operations. To adjust the head of operation c, one can build the JPS schedule where operation c has an infinite priority, thus operation c will start at time r_c. If the obtained schedule length is bigger than the upper bound *UB* of the HFS problem then the head of operation c can be increased. Let a_i^+ be the residual processing time of operation i at time r_c in the modified JPS schedule. Take

the operations of $K_c^+ = \{i | a_i^+ > 0, q_i > q_c\}$ in increasing order of q_i and find the first operation s for which relation $r_c + a_c + \sum_{q_i \geq q_s} a_i^+ + q_s > UB$ is satisfied. If such an operation exists then $r_c = \max(r_c, \max_{q_i \geq q_s} C_i)$ where C_i is the completion time of operation i in the usual JPS (where job c does not have an infinite priority). See [13] for more information and for an $O(n\log n)$ algorithm for updating the heads of all operations. Similarly the tails of operations can be adjusted by interchanging the roles of heads and tails.

3.2.5. Energetic reasoning

The previous constraints do not fully consider the limited number of machines at the second stage. In order to do so, we use the so called *energetic reasoning* in lower bound calculation for the multiprocessor scheduling problem [14–16].

Let $d_i = UB' - q_i^{II}$ be the deadline of the second stage operation i, where UB' represents an attempt of upper bound for the HFS problem. Given a time interval $[t_1, t_2] \subseteq [0, UB']$ we calculate for each job i the *left-work* $W_{left}(i, t_1, t_2)$ and the *right-work* $W_{right}(i, t_1, t_2)$ which represent the part of operation i that must be processed between t_1 and t_2 when the operation starts as soon as possible, thus at time r_i^{II}, and, respectively, as late as possible, at time $d_i - b_i$. The mandatory amount of work for operation i over the interval $[t_1, t_2]$ is the minimum between its left-work and right-work:

$$W(i, t_1, t_2) = \min(W_{left}(i, t_1, t_2), W_{right}(i, t_1, t_2))$$

The total amount of work for interval $[t_1, t_2]$ is the sum of works $W(i, t_1, t_2)$ for all the operations:

$$W(t_1, t_2) = \sum_i W(i, t_1, t_2)$$

If the total amount of work $W(t_1, t_2)$ exceeds the amount of available "energy" $m(t_2 - t_1)$ then the problem is infeasible. This property can be used to increase the global lower bound value. Let L be the best value of $GLB2$ obtained so far. Set $UB' = L$ and do the above computations. If an interval $[t_1, t_2]$ for which the problem is infeasible is found then the current UB' value can be increased by at least:

$$\Delta(t_1, t_2) = \left\lceil \frac{W(t_1, t_2) - m(t_2 - t_1)}{m} \right\rceil$$

The UB' value is adjusted by adding it to the maximal increase calculated for each time interval, the new value of UB' becomes a lower bound to the HFS problem:

$$UB' = L + \max_{[t_1, t_2] \subseteq [0, L]} (\Delta(t_1, t_2), 0) \tag{4}$$

The direct calculation of maximal increase using relation (4) is pseudo-polynomial because the number of time intervals that must be examined is proportional to L^2. Hopefully not all the intervals are relevant, in [14] it is proved that only $O(n^2)$ increase calculations are representative. Particularly, in a simplified version, only the intervals $[t_1, t_2]$, such that $t_1 \in \{r_i\} \cup \{r_i + b_i\} \cup \{d_i - b_i\}$ and $t_2 \in \{d_i\} \cup \{r_i + b_i\} \cup \{d_i - b_i\}$, have to be examined.

The available energy can also be used to calculate time-bound adjustments for operations release dates and deadlines. Let $SL(i, t_1, t_2) = m(t_2 - t_1) - W(t_1, t_2) + W(i, t_1, t_2)$ be the available energy over $[t_1, t_2]$ when operation i is not considered. If the left-work $W_{left}(i, t_1, t_2)$ of an operation i is bigger than the available energy $SL(i, t_1, t_2)$, then only a part, smaller or equal to $SL(i, t_1, t_2)$, of i can be processed during the interval $[t_1, t_2]$. The release date of operation i can be updated: $r_i = t_2 - SL(i, t_1, t_2)$. Similarly if $W_{right}(i, t_1, t_2) > SL(i, t_1, t_2)$ then the deadline is adjusted $d_i = t_1 + SL(i, t_1, t_2)$.

3.2.6. GLB2 computation

The computation of the global lower bound $GLB2$ is performed as follows. First, the inter-stage precedence, jobs precedence and cumulative previous work constraints are grouped into a constraint programming model and a constraint propagation method is used to compute the heads and the tails for each operation. The heads and tails obtained in this way are used in a list scheduling heuristic (defined in the sequel) in the priority function calculation. The solution found by the list scheduling is an upper bound, UB, for the HFS problem, which afterwards together with the JPS and energetic constraints are added to the constraint programming model defined above. Using the propagation technique new and eventually better values for operations heads and tails are obtained.

Due to the use in the JPS and energetic constraints of an upper bound, it is clear that the more this upper bound is tight the more the $GLB2$ is constrained, thus potentially better values for $GLB2$ could be obtained. The last fact motivated us to find an upper bound candidate UB', $UB' \in [GLB2, UB]$, such that for UB' the HFS problem is feasible and for $UB' - 1$ the problem becomes infeasible. A dichotomization procedure is introduced in order to explore the $[GLB2, UB]$ interval more optimally. In this way, a new global lower bound $GBL2^{dich} = UB'$ is obtained. The calculation of this bound is pseudo-polynomial and depends on initial UB' limits. In our calculation experiments the dichotomization procedure takes, in the worst case, less than 10 s, taking into account that the optimization of the constraint propagation code was not envisaged.

4. List scheduling

A reliable heuristic from the multiprocessor scheduling literature is the list scheduling (LS). Roughly speaking, in a LS algorithm the tasks are ordered (statically or dynamically) according to a priority rule and then are assigned in this order to the first available processor. Different priority rules have been proposed. Critical path based rules are known to provide the best results in the context of multiprocessor scheduling.

Algorithm 3 is a modified version of the LS heuristic which is used for solving the HFS problem. The main difference from the list scheduling used in multiprocessor problems is that in this algorithm the start time of a job takes into account also the first stage processing. The following notation are used in the algorithm: T is a variable that stores the time when the first stage machine is available (initially it is available at instant zero). When a second stage machine M is chosen for the current job to be scheduled we denote by F the moment of time when it is available. The only difference between the list scheduling we propose for the classical and for the no-wait HFS consists in how the update of the first stage machine availability time T is made (algorithm line 10).

Algorithm 3. List scheduling (LS) algorithm for the HFS problem (s_j – second stage start time of job j).

1: $S = \{0\}$ {Jobs ready for scheduling}
2: $s_0 = 0$
3: $T = 0$
4: **while** $S \neq \emptyset$ **do**
5: Calculate priorities p_i for jobs $i \in S$
6: Choose top priority job $j = \text{argmax}_{i \in S} p_i$, $S = S - j$
7: Choose the earliest available second stage machine M for j
8: Determine time F when machine M is available
9: Schedule j on M at time $s_j = \max(T + a_j, F)$
10: Classical HFS: $T = T + a_j$. No-wait HFS: $T = s_j$

11: $S = S \cup \{i \in \text{succ}(j)\}$ such that all the predecessors of i are finished
12: **end while**

Two priority rules are proposed. The first priority rule P^I is critical path based, particularly the CP/MISF (critical path/most immediate successors first) rule described in [17]. Priority value (5) is computed for each job $i \in S$ and the job with the largest p_i^I is chosen for being scheduled next:

$$p_i^I = q_i^{II} + \frac{|\text{succ}(i)|}{n+1} \tag{5}$$

This priority function ensures that the next job to schedule is the one which has the largest tail. Or, when the tails of two jobs are equal, it selects the job with the largest number of successors.

A second rule P^{II} is proposed because the critical path based rule does not take into account the idle time a list scheduling algorithm potentially creates at the first stage. In this priority rule, the next job to schedule is the one that *fits the best* the first stage machine free time, i.e. the job $i \in S$ having the highest value (6) is chosen for scheduling (here we use the same notation as in Algorithm 3):

$$p_i^{II} = -|F-(T+a_i)| \tag{6}$$

This priority rule has similarities with the ETF (earliest time first) rule from the multiprocessor scheduling [18], and actually when relation $T+a_i \le F$ is satisfied, P^{II} is the ETF priority rule.

5. Adaptive randomized list scheduling

A drawback of the list scheduling heuristic is that it returns a single solution by breaking any ties in the priority value of two or more jobs arbitrarily. Bad decisions in choosing the job to schedule (among the jobs having same priority), potentially, makes the heuristic to find low quality solutions on some instances. In order to overcome this drawback, the list scheduling algorithm can be executed several times, each time breaking ties randomly.

Inspired by the work [19] on the randomization of greedy algorithms, we further generalize this method by introducing a randomization parameter α, $\alpha \in [0,1]$, which aims to control the randomness of the list scheduling. Let S be the set of ready jobs to be scheduled and let $p_{max} = \max_{i \in S} p_i$, $p_{min} = \min_{i \in S} p_i$ be the maximum, respectively, the minimum priority values of these jobs. At each iteration of the list scheduling algorithm the next job to schedule is chosen uniformly from the jobs with the priorities belonging to the range $[p_{max}-\alpha(p_{max}-p_{min}), p_{max}]$. In this way, by adjusting coefficient α different behaviors of the list scheduling can be obtained, i.e. for $\alpha = 0$ we have the list scheduling with random ties breaking and for $\alpha = 1$ we obtain a list scheduling with a random priority rule.

The randomized list scheduling algorithm consists in executing the list scheduling with the random selection rule described above for a number of times and to retain the best obtained schedule as solution.

During the experimental phase a drawback of the randomized list scheduling was revealed. Actually the randomization parameter α cannot be chosen unequivocally for different problem parameters, as number of jobs, stage workloads, etc. The adaptive randomized list scheduling (ARLS) algorithm is then introduced to overcome this issue, see Algorithm 4. In this algorithm a preliminary phase is performed, during which the quality of solutions obtained for each randomization parameter is estimated. Thus, the randomized list scheduling is executed the same number of times *SampCnt* for each $\alpha \in \mathcal{A}$, where $\mathcal{A}$ is the set of used randomization parameters and the best solution S_α is saved. Afterwards, in function of the distance of S_α from the worst solution S_{max} obtained so far a proportional quota N_α from the total iteration count *IterCnt* is assigned to parameter α. Thus, better is the solution S_α more iterations with parameter α are done in the second phase. When all the solutions are equal the total iteration count is split into equal parts for each α. Finally, the randomized list scheduling is executed for each α, N_α iterations and the best obtained solution is returned.

Algorithm 4. Adaptive randomized list scheduling (ARLS).

Require: $\mathcal{A}$ - randomization parameters α to use
Require: *SampCnt* - number of sample runs for each α
Require: *IterCnt* - number of iterations for the search phase
Ensure Best found solution, *best*
$S_\alpha = RandomizedListScheduling(\alpha, SampCnt)$, $\forall \alpha \in \mathcal{A}$
$S_{max} = \max_\alpha S_\alpha$
$S_{min} = \min_\alpha S_\alpha$
if $S_{max} \neq S_{min}$ **then**
  $P_\alpha = (S_{max}-S_\alpha)/\sum_{\alpha'}(S_{max}-S_{\alpha'})$, $\forall \alpha \in \mathcal{A}$
else
  $P_\alpha = 1/|\mathcal{A}|$, $\forall \alpha \in \mathcal{A}$
end if
$N_\alpha = P_\alpha \cdot IterCnt$, $\forall \alpha \in \mathcal{A}$
$best = S_{min}$
for all $\alpha \in \mathcal{A}$ **do**
  $sol = RandomizedListScheduling(\alpha, N_\alpha)$
  if $best > sol$ **then**
    $best = sol$
  end if
end for
return *best*

The performance of ARLS relies on the good choice of sampling phase number of iterations *SampCnt*, on the randomization parameters α and on the second phase iterations count *IterCnt*. The parameter *SampCnt* must give statistically reliable estimates of P_α. In order to control the overall complexity of the ARLS algorithm the number of second iterations *IterCnt* shall be carefully chosen.

We must note that there is practically no use of adapting online the randomization parameter α. An ARLS version which is updating α during the execution was tested, the differences in the obtained solutions were negligible.

6. Experimental results

The algorithms described earlier were implemented using the C++ language. We have used the constraint propagation framework from ILOG CP solver to implement the *GLB2* calculation. The dichotomization procedure of $GLB2^{dich}$ calculation was implemented as a goal for CP solver. We shall note that only the constraint propagation feature of ILOG CP solver was used. The test programs were executed on an Intel Core2 Duo P8600 system without explicit parallelization.[1]

6.1. Instance generation

For testing the performance of the proposed heuristics and global lower bounds we use a set of 360 graphs from the

[1] As a multi-start heuristic, our algorithm can be straightforwardly parallelized.

"standard task graph set", which can be found in [20]. One half of the graph instances contain 50 jobs, the other half has 100 jobs. The graphs have either fully random structure or are composed of layers of random sizes. Each task processing time is randomly sampled using uniform, exponential or normal distributions with either one or two modes.

A HFS instance from such a graph is generated as follows. The precedence relations between the tasks are used as precedence relations for second stage operations.

The processing time c_i of task i is split into two parts, $a_i=\rho c_i$ and $b_i=(1-\rho)c_i$. Values a_i and b_i are rounded to the nearest integers such that relation $c_i=a_i+b_i$ remains valid. The coefficient ρ is used to obtain different load balancing between the first stage and the second stage. Let $r=\sum a_i/(\sum b_i/m)$ denote the desired ratio between the first and second stage workload (i.e. when $r=1$ the processing load is balanced between the stages). Then the coefficient ρ can be computed using relation:

$$\rho=\frac{r}{r+m}$$

Three ratios r are used in order to examine the performance of heuristics and of lower bounds for different load balances between the stages. For each task graph several HFS instances are generated, so 180 different HFS instances are obtained for each number of jobs, load ratio and number of the second stage machines.

6.2. Global lower bounds

In the first experiment we examine the relative performance of the global lower bounds for each version of HFS problem, the classical and the no-wait one. The global lower bounds *GLB*1, *GLB*2 and $GLB2^{dich}$ are computed for 9720 problem instances generated as described earlier, with $r\in\{\frac{2}{3},1,\frac{3}{2}\}$, $m\in 2,\ldots,10$ and $n=50, 100$.

First we want to study the improvement brought by the dichotomization procedure on the *GLB*2 quality. The instances for which the lower bound calculated by dichotomization, $GLB2^{dich}$, is strictly better than *GLB*2 are counted. For the classical HFS the dichotomization improved the lower bound of 1561 problem instances, which represents 16% from the total number. In the case of no-wait HFS the improvement was observed in 4355 (45%) cases. For instances of 100 jobs the number of improvements decreases slightly ($<1\%$) when compared to instances of 50 jobs. In order to sample the quality of these improvements the deviations $1-GLB2/GLB2^{dich}$ were calculated for each instance. In the case of classical HFS the average deviation is less than 0.1% and for the no-wait HFS less than 0.5%. Although the dichotomization procedure improves the *GLB*2 bound quality, its relatively high computation cost limits its use.

In a second experiment we study the relative performance of *GLB*1 and $GLB2^{dich}$. The number of instances for which *GLB*1 is strictly better, both bounds give the same value and $GLB2^{dich}$ is strictly better are counted. The results in percentage from the total number of instances are presented in Table 1. As we can observe there is no substantial difference in the behavior of bounds for classic and no-wait HFS, probably because the same set of instances is equally difficult for $GLB2^{dich}$ in both HFS types. Another interesting fact is that the quality of *GLB*1 increases for instances with more jobs. The result changes for "layered" instances, for which the *GLB*1 is strictly better than (equal to) $GLB2^{dich}$ in 13% (27%) of the cases for instances of 50 jobs and 17% (34%) for instances of 100 jobs no matter the HFS type.

In order to compare the performance of lower bounds function of load ratio and number of the second stage machines, for each pair (r,m) we count the number of times each global lower bound is strictly better than the other bound. Let $p_1(r,m)$ and $p_2(r,m)$ be the ratio the first bound is better $GLB1>GLB2^{dich}$ and, respectively, the second is better $GLB1<GLB2^{dich}$ expressed in percents from the total number of instances for each (r,m) and let $p_{1,2}(r,m)$ be the ratio the bounds are equal. In Fig. 4 $p_1(r,m)$, $p_{1,2}(r,m)$ and

Table 1
Relative comparison of *GBL*1 and $GLB2^{dich}$. The percentages of instances for which $GLB1>GLB2^{dich}$, $GLB1=GLB2^{dich}$ and $GLB1<GLB2^{dich}$ are presented.

HFS type	n	*GLB*1 vs. $GLB2^{dich}$		
		$>$	$=$	$<$
Classic	50	8.58%	23.17%	68.25%
	100	11.67%	26.98%	61.36%
No-wait	50	8.48%	22.16%	69.36%
	100	11.63%	26.11%	62.26%

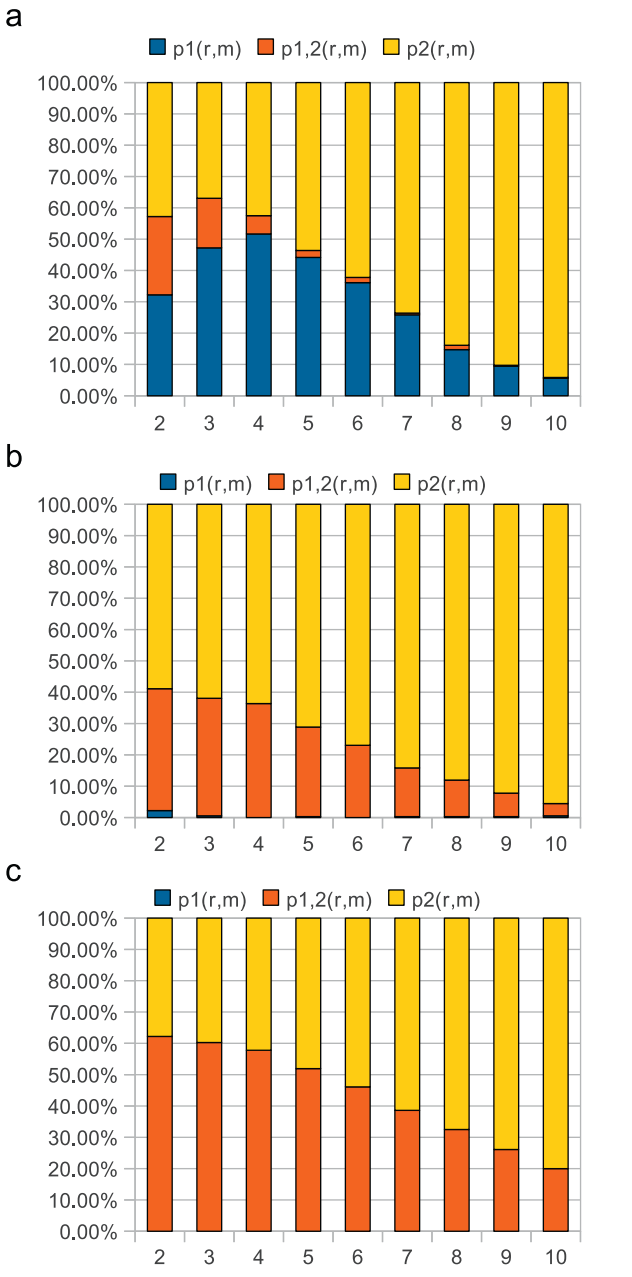

Fig. 4. Relative comparison of *GLB*1 and $GLB2^{dich}$ for each pair (r,m) of parameters. (a) $r=\frac{2}{3}$. (b) $r=1$. (c) $r=\frac{3}{2}$.

$p_2(r,m)$ are plotted for each pair (r,m). The results for classic HFS and no-wait HFS are practically the same, consequently only the no-wait case is plotted.

We observe that for load ratios $r=1$ and $\frac{3}{2}$ the $GLB1$ bound is practically never greater than $GLB2^{dich}$. For small number of the second stage machines m the first bound performs better than for large m, being equal to $GLB2^{dich}$ in approximatively 60% of the cases for $r=\frac{3}{2}$ and 40% for $r=1$ when $m=2$. We suppose that this is due to the fact that for instances with large first stage workloads the one-machine based constraints perform better than the simple sum of the first stage durations, used in $GLB1$.

When the second stage workload is dominating, $r=\frac{2}{3}$, the first global lower bound performs better than in the previous cases. The best results of $GLB1$ are obtained for $m=4$ the first bound being better in more than 50% of the cases. For other values of the second stage machines, lesser or greater than 4, the performance of $GLB1$ decreases. The definition of $GLB1$ makes it perform better on instances where the workloads are asymmetrically distributed between the stages. This can be seen in the results, the overall performance of $GLB1$ is lower for $r=1$ than for other two load ratios.

For all load ratios, with the increase of number of machines m the relative performance of the second global lower bound also increases, obviously due to the fact that for large values of m the second stage critical path plays a higher role in the HFS execution.

6.3. List scheduling heuristics

Firstly we shall investigate the influence of sampling phase iterations count *SampCnt* and second phase number of iterations *IterCnt* on the ARLS performance. The goal is to choose parameters that produce good solutions of the heuristic when compared to its complexity. The same set of instances as in previous section is used.

The randomization parameter α takes five values from the set $\mathcal{A}\in\{0,0.2,0.4,0.6,0.8\}$. As the fully randomized list scheduling is outside the scope of this study, value 1 for parameter α is not used.[2] Theoretically, when $\alpha=1$ the used scheduling priority rule should not influence the results because the list scheduling is fully random. A finer division for α is not necessary because the performance increases insignificantly, but the total number of iterations raises.

In the sampling phase of ARLS heuristic six values of iterations count *SampCnt* have been tested: 50, 100, 150, 200, 250 and 300. In order to have the same total number of iterations independently of *SampCnt* value the number of second phase iterations count is $IterCnt=n^{\beta}+(300-SampCnt)\cdot|\mathcal{A}|$, where $1.5\le\beta\le 2$. Six values for β are experimented with such that the obtained *IterCnt* are equidistantly situated. The execution time, in the worst observed case, is under 5 s and mainly depends on the graph edge density.

In order to minimize the influence of the randomness on the performance study, for each problem instance the ARLS algorithm is executed 10 times and the averaged result is retained for comparison. The deviation $S/GLB_{max}-1$ of the averaged solution S from the maximal global lower bound $GLB_{max}=(GLB1,GLB2^{dich})$ is calculated for each instance.

A preliminary experiment has proved that better solutions are obtained when the ARLS heuristic is executed two times, first with priority rule P^{I} and after with P^{II}, keeping the best solution of each run, even if the total number of iterations is two times smaller.

[2] We have executed the ARLS heuristic also with $\alpha=1$ but no increase in the quality of the solutions was observed.

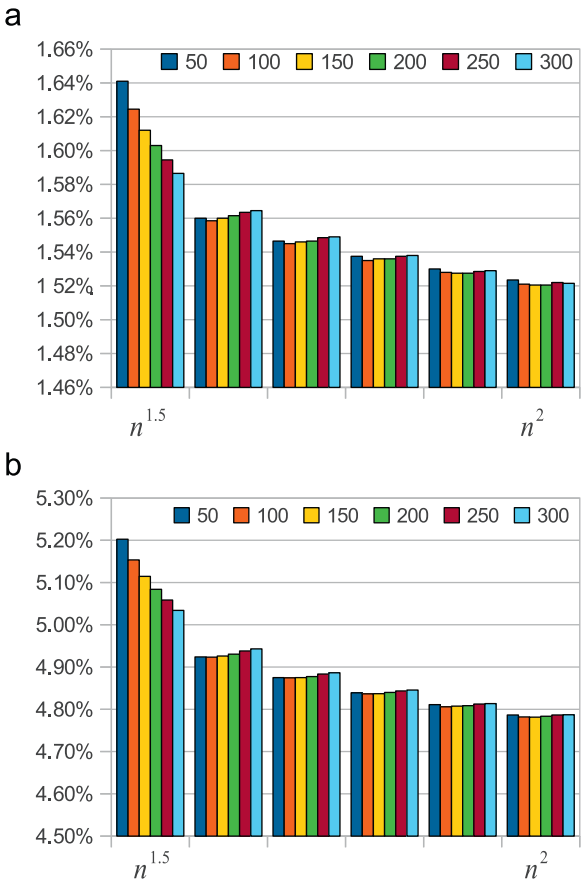

Fig. 5. Influence of parameters *SampCnt* and *IterCnt* on the average deviation of the ARLS algorithm (the *IterCnt* parameter is on horizontal axis and different bar colors represent *SampCnt*). The deviation is computed for the minimal solution obtained by the two priority rules. (a) Classical HFS. (b) No-wait HFS. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Upper bound on the average deviations established by algorithms ARLS and LS together with GLB_{max}.

HFS type	50		100	
	LS (%)	ARLS (%)	LS (%)	ARLS (%)
Classic	2.98	1.83	1.95	1.21
No-wait	7.83	4.62	7.97	4.94

The averaged deviations for each *SampCnt* and *IterCnt* are illustrated in Fig. 5. We observe that when the second phase iterations count is the lowest $IterCnt=n^{1.5}$, better results are obtained for larger sampling phase iterations count. This can be explained by the fact that the second phase iterations number is insufficient in order to explore the solution space. Another interesting fact is that for large second phase iterations count it is not always better to have larger sampling phases. We suppose this is due to the fact that parameter P_{α} (see Algorithm 4) is reliably enough estimated for smaller *SampCnt* and it is better to do more iterations in the second phase. For $IterCnt=n^2$ the difference in solutions obtained with different *SampCnt* is insignificant, being under 0.01%. It can be seen that a sampling phase with $SampCnt=100$ gives statistically reliable estimations for the parameter P_{α}. Note that, contrary to *IterCnt*, *SampCnt* can be chosen independently of the instance size, based on statistical convergence considerations. So we use this sample phase

Table 3
Average deviation of the minimal solution found by the ARLS heuristic using both priority rules.

m	n							
	50				100			
	$r=\frac{2}{3}$ (%)	$r=1$ (%)	$r=\frac{3}{2}$ (%)	$\overline{dev_m}$ (%)	$r=\frac{2}{3}$ (%)	$r=1$ (%)	$r=\frac{3}{2}$ (%)	$\overline{dev_m}$ (%)
(a) Classical HFS								
2	0.47	3.23	0.24	1.31	0.24	2.16	0.07	0.82
3	0.82	3.41	0.48	1.57	0.47	2.36	0.31	1.05
4	1.17	3.29	0.74	1.73	0.55	2.31	0.78	1.21
5	1.50	2.99	1.54	2.01	0.67	2.16	1.05	1.29
6	1.64	2.72	2.37	2.25	0.77	1.79	1.33	1.30
7	1.63	2.25	3.07	2.32	0.82	1.63	1.80	1.42
8	1.20	1.82	3.17	2.07	0.65	1.26	2.08	1.33
9	0.78	1.29	3.14	1.73	0.53	1.01	2.29	1.28
10	0.45	0.92	3.03	1.46	0.33	0.86	2.50	1.23
$\overline{dev_r}$(%)	1.07	2.44	1.98	**1.83**	0.56	1.73	1.36	**1.21**
(b) No-wait HFS								
2	2.53	10.18	2.59	5.10	2.80	11.56	3.32	5.89
3	3.14	9.67	2.32	5.04	3.09	10.64	2.72	5.49
4	3.53	8.94	2.78	5.08	3.09	10.07	3.43	5.53
5	3.61	7.75	3.56	4.97	3.21	9.35	3.84	5.47
6	3.69	7.03	4.52	5.08	3.09	8.21	4.21	5.17
7	3.49	6.03	5.28	4.93	2.83	7.13	4.79	4.92
8	2.66	4.97	5.44	4.36	2.34	5.93	5.00	4.42
9	1.86	3.81	5.59	3.75	1.86	4.82	5.24	3.98
10	1.22	2.96	5.69	3.29	1.32	3.96	5.58	3.62
$\overline{dev_r}$(%)	2.86	6.81	4.20	**4.62**	2.62	7.96	4.24	**4.94**

iterations number in the next experiments, n^2 is used for second phase iterations count.

In the next experiment the priority rules are compared. It was determined that for the classical HFS the *PI* priority rule dominates in average *PII* in all the test instances, which can be explained by the dominance of the multiprocessor scheduling problem in the classical HFS, for which critical path rules are better. In the case of no-wait HFS the second priority rule *PII* produces better solutions for load ratios $r=\frac{2}{3}$ and 1 and for a second stage machines count $m \leq 4$.

In order to see the improvement of the randomization on the ordinary list scheduling in Table 2 the quality of solutions obtained by the ARLS heuristic and the ordinary list scheduling heuristic are compared. The randomization always improves the solutions found by the list scheduling, the deviations of solutions are decreased by ARLS with approximatively 40%.

Table 3 presents the average deviations of the solutions calculated by the ARLS heuristic in function of work load ratio r, second stage machines count m and number of jobs n. Also in the table are illustrated the averaged values of deviations for each m and r. As we can see on average for the classical HFS the deviation is lower than 2% and for the no-wait case the deviation is under 5%. The deviations per number of the second stage machines, $\overline{dev_m}$, tend to decrease for larger values of m. In both HFS types the hardest instances are those for which the workload is balanced between the stages, i.e. $r=1$, the largest deviations being obtained for small m. In the case of no-wait HFS, when $m=2$ and $r=1$ the deviation is less than 11% for 50 jobs and less than 12% for 100 jobs. With the increase of the number of the second stage machines this deviance decreases, being under 4% for $m=10$. A closer examination revealed that the largest deviations are obtained for instances for which the processing times distributions follow an exponential law. In order to see what is their influence, the deviations were recalculated without the exponential processing time instances. It was found that in the case of no-wait HFS the worst observed deviation falls down from 12% to 8%.

7. Conclusion

In this study two versions of the two-stage hybrid flow shop problem with second stage precedence constraints and parallel machines are investigated, the classical and the no-wait one. An adaptive randomized list scheduling (ARLS) heuristic, together with two priority rules, is proposed for solving both problem versions. Our heuristic is made of a constructive part (ARLS) associated to a global lower bound which allows to obtain provably good solutions.

The practical application of the hybrid flow shop problem occurs in the scheduling of an algorithm on a parallel computer, where the memory accesses are independent from task execution. Using this problem a more fine-grained modeling of an on-line execution of an algorithm on a parallel computing system can be obtained.

The evaluation of the heuristic is done using randomly generated problem instances. The ARLS algorithm gives better schedules for all of the examined cases when compared to the ordinary list scheduling. The best results are obtained in the case of the classical HFS problem version, with an average deviation established by the algorithm under 2% from the optimum. For the no-wait HFS version that deviation is smaller than 5%. The critical path based priority rule provides better solutions in average.

The fact that the randomization increases the quality of the list scheduling solutions motivates us to examine, in a subsequent work, a more complex probabilistic heuristic, e.g. by introducing a local search phase in a GRASP-like fashion.

In future works we plan to investigate more general hybrid flow shop models, with parallel machines on both stages so as to take into account several memory access channels in a parallel computer architecture, or, with stochastic job processing times in order to take into account the incertitude of task durations.

References

[1] Ruiz R, Şerifoğlu F, Urlings T. Modeling realistic hybrid flexible flowshop scheduling problems. Computers & Operations Research 2008;35(4): 1151–75.

[2] Ribas I, Leisten R, Framinan J. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research 2010;37(8):1439–54.
[3] Gupta J, Hariri A, Potts C. Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. Annals of Operations Research 1997;69: 171–91.
[4] Gladky A, Shafransky Y, Strusevich V. Flow shop scheduling problems under machine-dependent precedence constraints. Journal of Combinatorial Optimization 2004;8(1):13–28.
[5] Strusevich V. Shop scheduling problems under precedence constraints. Annals of Operations Research 1997;69:351–77.
[6] Guinet A, Legrand M. Reduction of job-shop problems to flow-shop problems with precedence constraints. European Journal of Operational Research 1998;109(1):96–110.
[7] Dror M, Mullaseril P. Three stage generalized flowshop: scheduling civil engineering projects. Journal of Global Optimization 1996;9:321–44. (24).
[8] Botta-Genoulaz V. Considering bills of material in hybrid flow shop scheduling problems. In: IEEE international symposium on assembly and task planning 1997, ISATP 97; 1997. p. 194–9.
[9] Botta-Genoulaz V. Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. International Journal of Production Economics 2000;64(1–3):101–11.
[10] Gupta J. Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society 1988;39:359–64.
[11] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. New York, NY, USA: W.H. Freeman & Co.; 1979.
[12] Carlier J. The one-machine sequencing problem. European Journal of Operational Research 1982;11(1):42–7.
[13] Carlier J, Pinson E. Adjustment of heads and tails for the job-shop problem. European Journal of Operational Research 1994;78(2):146–61.
[14] Baptiste P, Le Pape C, Nuijten W. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research 1999;92:305–33.
[15] Lopez P, Erschler J, Esquirol P. Ordonnancement de tâches sous contraintes: une approche énergétique. Automatique, Productique, Informatique, Industrielle 1992;26:453–81.
[16] Lang T, Fernandez E. Improving the computation of lower bounds for optimal schedules. IBM Journal of Research and Development 1977;21(3):273–80.
[17] Kasahara H, Narita S. Practical multiprocessor scheduling algorithms for efficient parallel processing. IEEE Transactions on Computers 1984;33(11): 1023–9.
[18] Hwang J-J, Chow Y-C, Anger F, Lee C-Y. Scheduling precedence graphs in systems with interprocessor communication times. SIAM Journal on Computing 1989;18(2):244–57.
[19] Hart JP, Shogan AW. Semi-greedy heuristics: an empirical study. Operations Research Letters 1987;6(3):107–14.
[20] Standard task graph set, 〈http://www.kasahara.elec.waseda.ac.jp/schedule/index.html〉, last access January 8, 2011.