



**HAL**  
open science

# Improved Bounds for Hybrid Flow Shop Scheduling with Multiprocessor Tasks

Asma Lahimer, Pierre Lopez, Mohamed Haouari

► **To cite this version:**

Asma Lahimer, Pierre Lopez, Mohamed Haouari. Improved Bounds for Hybrid Flow Shop Scheduling with Multiprocessor Tasks. 2012. hal-00680452v1

**HAL Id: hal-00680452**

**<https://hal.science/hal-00680452v1>**

Preprint submitted on 19 Mar 2012 (v1), last revised 3 Sep 2013 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Improved Bounds for the Hybrid Flow Shop Problem with Multiprocessor Tasks

Asma LAHIMER,<sup>1</sup> Pierre LOPEZ<sup>2,3,\*</sup> and Mohamed HAOUARI<sup>4,5</sup>

<sup>1</sup>Department of Mathematics and Computer Science, INSAT, University of Carthage, Tunisia

<sup>2</sup>CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France

<sup>3</sup>Univ de Toulouse, LAAS, F-31400 Toulouse, France

<sup>4</sup>Department of Industrial Engineering, Ecole Nationale d'Ingénieurs de Tunis, University of Tunis El Manar, Tunisia

<sup>5</sup>Department of Industrial Engineering, Ozyegin University, Istanbul, Turkey

**Abstract.** We investigate the problem of minimizing makespan in a multi-stage hybrid flow shop with multiprocessor tasks. To generate high-quality approximate solutions to this challenging  $\mathcal{NP}$ -hard problem, we propose a new discrepancy search method that is based on adjacent discrepancies. Furthermore, we describe a new lower bound that is based on the concept of dual feasible functions. The proposed lower and upper bounds are assessed through computational experiments on 300 benchmark instances with up to 100 jobs and 8 stages. For these instances, the proposed lower and upper bounds consistently outperform the best existing ones. In particular, the proposed heuristic successfully improved the best known solution of 75 benchmark instances.

**Key words:** Hybrid flow shop scheduling, Multiprocessor tasks, Discrepancy search, Dual feasible functions.

## 1. Introduction

The Hybrid Flow Shop scheduling problem with multiprocessor tasks can be formally described as follows: A set  $J = \{1, 2, \dots, n\}$  of  $n$  jobs, have to be processed in a manufacturing system with  $m$  production stages (or, centers). Each stage  $i \in M = \{1, 2, \dots, m\}$  consists of  $m_i$  identical parallel processors. Each job  $j \in J$  has to be processed non-preemptively on stages  $1, 2, \dots, m$  in that order. That is, all jobs serially traverse stages following the same production route (see Figure 1). For processing job  $j \in J$  in

---

\* *email:* pierre.lopez@laas.fr

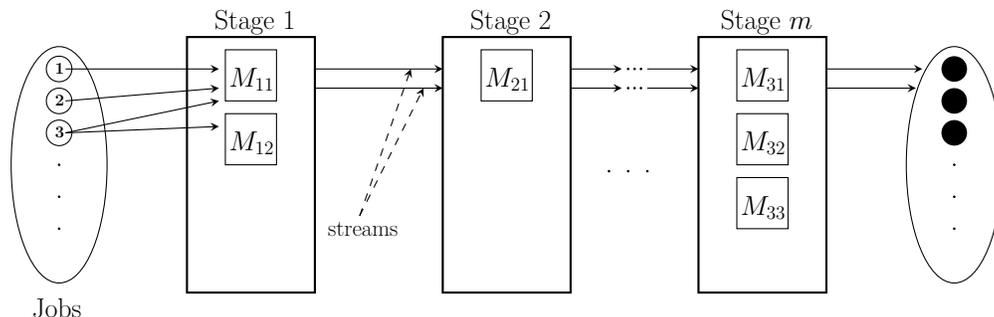


Figure 1: A hybrid flow shop with multiprocessor tasks

stage  $i \in M$ ,  $size_{ij}$  processors are *simultaneously* required during  $p_{ij}$  units of time. For example, we see in Figure 1 that Job 3 requires two machines in Stage 1. Hence, unlike classical scheduling models where it is usually assumed that a task (operation) requires for its processing only one processor at a time, we consider that  $J$  includes multiprocessor tasks, each of which may require a set of processors at a time, what is generally called “parallel processor requirement” [11]. The objective is to minimize the makespan ( $C_{\max}$ ), that is, the completion time of all jobs in the last stage. Using the classical 3-field notation in production scheduling, the problem is denoted by  $FHm, ((PM^{(k)})_{k=1}^m) |size_{ij}| C_{\max}$ . Applications of this scheduling problem with multiprocessor tasks can be found in various manufacturing systems (*e.g.*, work-force assignment [7], transportation problems with recirculation [4]), printed circuit boards and semiconductors [9], as well as in some computer systems (*e.g.*, real-time machine-vision [12]).

Clearly, the special case where exactly one machine is required for each job in each stage (i.e.,  $size_{ij} = 1, \forall i \in M, j \in J$ ) reduces to the much studied hybrid flow shop problem (*HFS*). This latter scheduling problem has been intensively investigated in the scheduling literature. For updated and comprehensive state-of-the-art reviews, we refer to [29] and [30]. By contrast, and despite its practical importance, the multiprocessor task HFS variant has been only investigated in a relatively few papers, where classical meta-heuristic approaches have been tailored to solve this challenging scheduling problem. These contributions include: (*i*) Genetic Algorithms ([24], [25]), Serifoglu and Ulusoy [31], (*ii*) Tabu Search [26], (*iii*) Ant Colony Optimization [34], and (*iv*) Memetic Algorithms [17]. Furthermore, lower bounds have been proposed by Oğuz and Ercan [24].

In this paper, we make the following contributions:

1) We propose a new discrepancy-based search method called *Climbing Depth-bounded Adjacent Discrepancy Search* (CDADS) to solve the hybrid flow shop problem with multiprocessor tasks.

2) We propose a new lower bound that is based on the concept of dual feasible functions (DFFs).

3) We present computational results using a large set of benchmark instances that demonstrate the excellent performance of the proposed heuristic and lower bound. Indeed, we found that CDADS outperforms the best existing method yielding new improved solutions for a significant portion of the benchmark instances (25%). Moreover, we found evidence that the DFF-based lower bound consistently dominates state-of-the-art lower bounds.

The remainder of this paper is organized as follows. In Section 2, we introduce the principle and variants of discrepancy-based search methods. Next, we provide, in Section 3, a detailed description of the proposed approach. In Section 4, we present lower bounds from the literature as well as a new DFF-based lower bound. In Section 5, we report the results of an extensive computational study and analyze the performance of the proposed solution approach and lower bound. Finally, Section 6 provides some concluding remarks and directions for future research.

## 2. Discrepancy Search Methods

Limited discrepancy search (LDS) was introduced in 1995 by Harvey and Ginsberg [16]. This seminal method can be considered as an alternative to branch-and-bound, backtracking, and iterative sampling. From an optimization view-point this technique is similar to variable neighborhood search. Discrepancy search has been further extended in the literature ([14], [20]) to become Local Branching applied to Mixed-Integer Programs (MIPs) and Constraint Programming (CP). Parisini and Milano [27] recently introduced an improving CP-based local branching via sliced neighborhood search. The neighborhood in local branching is defined using the spirit of limited discrepancy search. Indeed, it starts from an initial global instantiation suggested by a given heuristic and successively explores branches with increasing discrepancies from it, in order to obtain a solution (in a satisfaction context), or a solution of better performance (in an optimization context). A discrepancy is associated with any decision point in a search tree where the choice goes against the heuristic. For convenience, in a tree-like representation the heuristic choices are associated with left branches while right branches are considered as discrepancies. Figure 2 illustrates the spirit of LDS. At the  $k^{th}$  iteration, solutions having discrepancies between 0 and  $k$  are visited. The

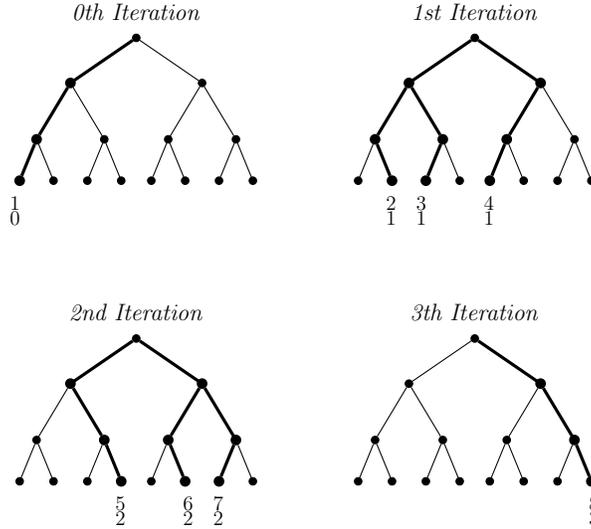


Figure 2: Improved Limited Discrepancy Search

first line in the figure illustrates the ordering in which the branches are visited while the second line shows the number of discrepancies associated with each leaf node. Since the inception of LDS, several variants have been proposed in the literature. In particular, we quote the following variants: Improved Limited Discrepancy Search (ILDS) [21], Depth-bounded Discrepancy Search [33], Discrepancy-Bounded Depth First Search [1] and Climbing Discrepancy Search (CDS) [23].

For the sake of brevity, we shall restrict to a concise description of two variants DDS and CDS that will subsequently serve as a basis for the development in Section 3.1 of a new variant.

**Depth-Bounded Discrepancy Search (DDS)** developed in [33], is an improved LDS that prioritizes discrepancies at the top of the tree to correct early mistakes first. This assumption is ensured by means of an iteratively increasing bound on the tree depth. Discrepancies below this bound are prohibited. DDS starts from an initial solution. At the  $i^{th}$  iteration, it explores those solutions on which discrepancies occur at a depth not greater than  $i$ .

**Climbing Discrepancy Search (CDS)** is a local search method adapted to combinatorial optimization problems proposed in [23]. CDS starts from an initial solution that would be dynamically updated. Indeed, it visits branches progressively until a better solution is reached. Then, the initial solution is updated and the exploration process is restarted.

### 3. Climbing Depth-Bounded Adjacent Discrepancy Search

#### 3.1 *Main Features*

To stick to the problem under consideration, we now consider an optimization context. We propose CDADS (Climbing Depth-bounded Adjacent Discrepancy Search) method, that is a combination of a depth-bounded discrepancy search and a climbing discrepancy search. We also assume that, if several discrepancies occur in the construction of a solution, these discrepancies are necessarily adjacent in the list of successive decisions.

CDADS starts from an initial solution obtained by a given heuristic, and explores its neighborhood progressively, according to the depth-bounded discrepancy search strategy. Hence, a limit depth  $d$  is fixed. Discrepancies below this bound are prohibited. At  $i^{th}$  iteration, we allow  $i$  discrepancies above the limit level  $d$ . Until this step, CDADS follows the same principle of Climbing Depth-bounded Discrepancy Search proposed in [2], [3] when solving the more classical hybrid flow shop scheduling problem. When considering solutions with more than one discrepancy, we require these discrepancies are achieved consecutively, that means a solution consists of discrepancies that happen one after the other. This assumption of adjacency considerably limits the search space. We also consider that the initial solution is generated by a heuristic. Thus, only the immediate neighborhood of a discrepancy may receive an additional discrepancy. Even if we are aware that other strategies for limiting the search space could be envisaged (focusing for example on given subsets of discrepancies), we bet that only performing adjacent discrepancies is promising. We then obtain a truncated DDS based on adjacent discrepancies, DADS (Depth-bounded Adjacent Discrepancy Search). This approach is illustrated by an example on a binary tree of depth 3 (see Figure 3). At the starting point, DADS visits the initial leaf node recommended by the heuristic. For convenience, we assume that left branches follow the heuristic. At first iteration, DADS visits leaf nodes at the depth limit with exactly one discrepancy. The first line shown under the branches reports the visit order of considered leaf node, while the second line illustrates the number of discrepancies made in each solution. The second iteration allows to exploring more leaves with two discrepancies with respect to the adjacency assumption. In this representation, the maximum depth bound is taken to be 3. If now, we limit the depth to two levels, several branches would not be retained, namely the leaf nodes 4, 6, and 7 would not be visited by DADS. Going back to the optimization issue, CDADS merges the DADS strategy with a CDS exploration principle, that is the initial solution used by DADS is dynamically updated when a best solution is found, and the exploration

process is restarted.

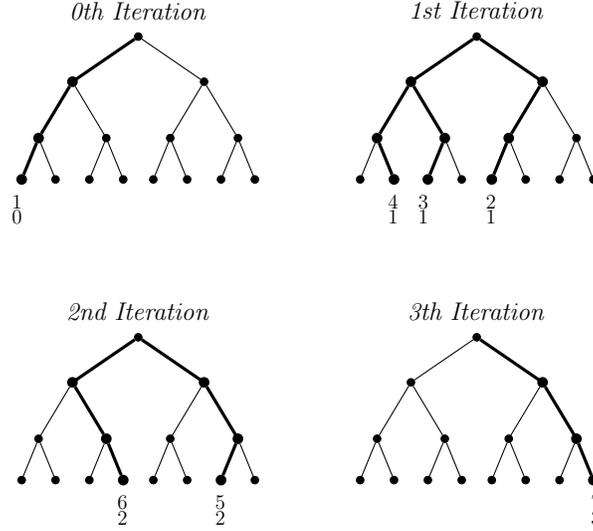


Figure 3: Depth-bounded Adjacent Discrepancy Search

## 3.2 Additional features

### 3.2.1 Heuristics

CDADS is strongly based on the quality of the initial solution. Thus, we carried out an experimental comparison between ten priority rules presented in the literature ([25], [34]). We considered the most effective heuristics to multiprocessor task hybrid flow shop scheduling. The four selected rules are:

**Shortest Processing Time (SPT)**, which ranks jobs according to the ascending order of their processing times;

**Shortest Processing Requirement (SPR)**, which ranks jobs according to the ascending order of their processing requirement;

**Energy rule (ER)**, considering first the jobs with the smallest energy (where the energy of an operation  $j$  at a stage  $i$  is evaluated by  $p_{ij} \times size_{ij}$ ;

**Normalized SPT** applied at the last stage (NSPT\_LS). For this latest rule, Ying and Lin [34] propose to schedule jobs according to their ranking index ( $RI_j$ ) defined by:  $RI_j = \frac{\max\{p_{mk}\} - p_{mj} + 1}{\max\{p_{mk}\} + 1}$ .

To assess the relative performance of these rules, we performed some preliminary experiments. The results are displayed in Table 1. This table reports, for each rule, its performance, *i.e.*, the ratio of best solutions obtained by the method.

Priority rule	Performance (%)
NSPT_LS	27
ER	25
SPT	17
SPR	14

### 3.2.2 Schedule Generation Scheme

Schedule generation schemes (SGSs) are widely used in solving preemptive problems. We distinguish between serial SGS and parallel SGS. These two heuristics ensure task scheduling based on a given priority rule. Hence, tasks are selected one after the other and a start time is fixed for each one. Serial SGSs are introduced in [19]. At each iteration, the first available task in  $\zeta$  is selected, where  $\zeta$  is the priority list recommended by the priority rule. The selected task is scheduled as soon as possible with respect to both resource constraints and precedence constraints. Parallel SGSs developed in [5], suggest a chronological procedure in scheduling tasks. At each time  $t$ , a set  $\zeta_t$  of tasks being scheduled is defined: this set contains unscheduled tasks that can be processed at  $t$  without breaking neither precedence constraints nor resource constraints. If we consider that  $\underline{t}$  is the first time where  $\zeta_{\underline{t}} \neq \emptyset$ , the first task in the priority list  $\zeta$  belonging to  $\zeta_{\underline{t}}$  is performed at  $\underline{t}$ . The same process is applied until all tasks are scheduled. The two schemes depicted above may appear similar. However, the schedules they generate are different: a serial SGS provides an active schedule while a parallel SGS generates a non-delay schedule. In the scheduling theory, Sprecher *et al.* [32] show that the set of active schedules includes at least one optimal solution. On the contrary, non-delay schedules may eliminate all optima. Concerning our method CDADS, we do not enumerate all possible solutions, so even serial SGSs may exclude all optimum solutions. Furthermore, in practice, parallel SGSs are known for their operational efficiency. Hence, we opt for the implementation of a parallel SGS which has been proved, moreover, to be more efficient in our experimental studies.

### 3.2.3 Encoding/Decoding Solution

An encoding of a solution to the  $FHm, ((PM^{(k)})_{k=1}^m) |size_{ij}| C_{\max}$  problem is to consider the sequence of jobs at each stage since different permutation of jobs may occur at different stages. In this representation, a solution will have  $i$  lists, each being a permutation of  $(1, 2, \dots, n)$ , corresponding to the job list at different stages. However, as explained by Oğuz and Ercan in [24], one can easily notice that we can only search for different permutations

of jobs at the first stage, and iteratively find the others lists on the remaining stages. Hence, we prefer to use only the sequence of jobs at the first stage in the encoding of a solution and then to decode each solution to a full schedule by applying a List Scheduling algorithm to process the jobs at other stages. We detail further the principle of the list algorithm considered for solutions decoding.

**Example 1:** Consider 7 jobs to be scheduled in a two-stage hybrid flowshop, with five processors in each stage. The processing times  $p_{ij}$  and the processor requirements of the jobs  $size_{ij}$  are given in Table 2.

Table 2: Data for Example 1

$j$	1	2	3	4	5	6	7
$p_{1j}$	1	4	2	1	1	2	2
$size_{1j}$	1	4	3	1	2	2	4
$p_{2j}$	2	2	1	2	2	2	3
$size_{2j}$	2	2	3	2	1	3	4

Assume that the selected priority rule produced the permutation  $(4, 3, 6, 7, 1, 2, 5)$ . The list algorithm will decode this sequence into a schedule as depicted in Figure 4. At Stage 1, the sequence is  $seq1 = (4, 3, 6, 7, 1, 2, 5)$  and we schedule the jobs by iteratively assigning them to the processors according to this order and to their processor requirements starting at time 0. After scheduling jobs 4 and 3, the next job to be scheduled is job 6. Because of the capacity constraint, job 6 cannot be scheduled earlier than time 1. At time 2, even though job 1 is available and there are enough processors, we can not schedule it because this will violate the precedence relation between 7 and 1 coming from the order in  $seq1$ . As a result both jobs 1 and 7 start at the same time. Similarly, after scheduling job 2 at time 5, since there are processors available for it, we schedule job 5 at time 9 as well. In the next step, we obtain the new sequence  $seq2$  to be followed on jobs scheduling at Stage 2. The permutation  $seq2$  is constructed by listing jobs in a non-decreasing order of their completion time at the previous stage. Thus, the new list  $seq2$  will be  $(4, 3, 6, 1, 7, 2, 5)$ . This illustrates that we do not limit to permutation schedules only ( $seq2$  is different from  $seq1$ ). We schedule jobs at Stage 2 according to  $seq2$  and by considering the completion time of jobs at Stage 1; that is, a job cannot start at Stage 2 before its completion time at Stage 1.

Implementing a parallel SGS works as follows: After scheduling jobs 4 and 3 satisfying their processor requirements at Stage 2, the next job to be scheduled is job 6 and since the number of available processors is enough for job 1, we schedule it at time 4 (end of job 1 at Stage 1). Similarly, we schedule the next job in list *seq2*, which is job 7, starting at time 6. We finally schedule job 2 and job 5, starting at times 9 and 10, respectively.

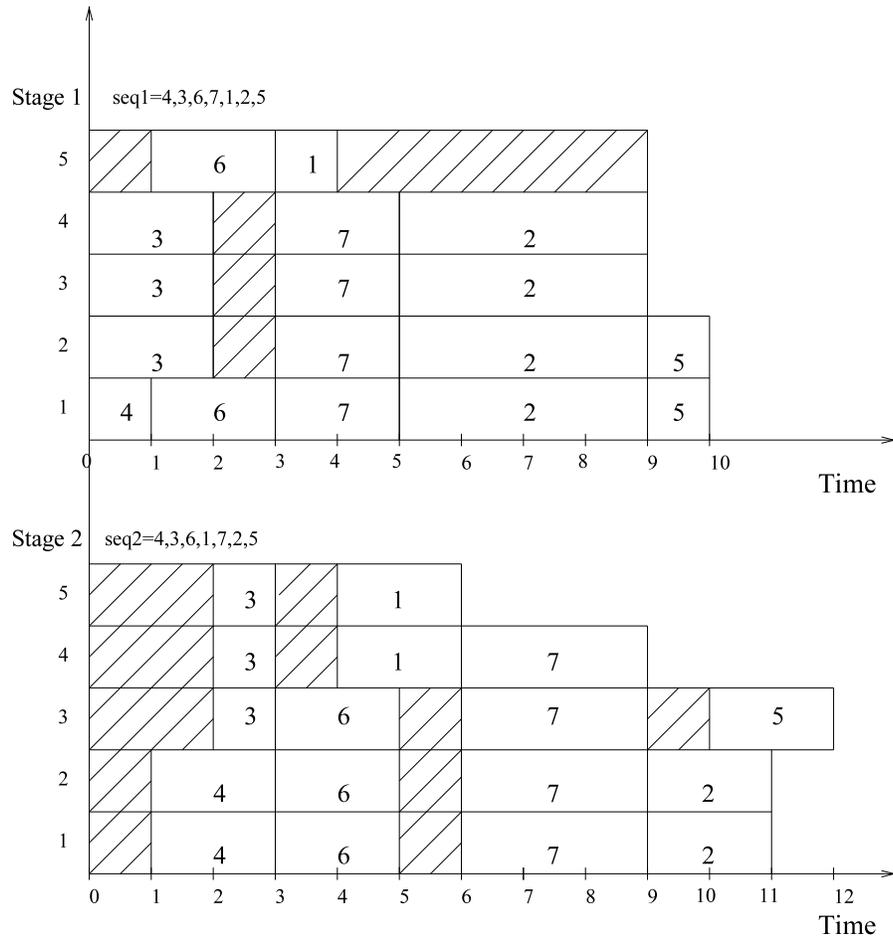


Figure 4: Gantt chart of the solution corresponding to Example 1

## 4. Lower bounds

### 4.1 Lower bounds from the literature

We define for each job  $j \in J$ , and each stage  $i \in M$  a head  $r_{ij}$  and a tail  $q_{ij}$ , that are computed by setting:

$$\left\{ \begin{array}{l} r_{ij} = \sum_{k=1}^{i-1} p_{kj} \text{ if } i > 1 \\ r_{ij} = 0 \text{ if } i = 1 \end{array} \right. , \text{ and } \left\{ \begin{array}{l} q_{ij} = \sum_{k=i+1}^m p_{ij} \text{ if } i < m \\ q_{ij} = 0 \text{ if } i = m \end{array} \right. ,$$

respectively. Also, we set:

$$\left\{ \begin{array}{l} A_i = \{j \in J : size_{ij} > \frac{m_i}{2}\} \\ B_i = \{j \in J : size_{ij} = \frac{m_i}{2}\} \end{array} \right. , \text{ for } i \in M.$$

Oğuz *et al.* [26] proposed the following simple lower bound:

$$LB_1 = \max_{i \in M} \left\{ \min_{j \in J} r_{ij} + \alpha_i + \min_{j \in J} q_{ij} \right\} \quad (1)$$

where

$$\alpha_i = \left\lceil \frac{1}{m_i} \sum_{j \in J} size_{ij} p_{ij} \right\rceil, \text{ for } i \in M.$$

Later, Oğuz and Ercan [24] introduced the following better lower bound:

$$LB_2 = \max_{i \in M} \left\{ \min_{j \in J} r_{ij} + \max\{\alpha_i, \beta_i\} + \min_{j \in J} q_{ij} \right\} \quad (2)$$

where

$$\beta_i = \left\lceil \sum_{j \in A_i} p_{ij} + \frac{1}{2} \sum_{j \in B_i} p_{ij} \right\rceil, \text{ for } i \in M.$$

The validity of  $LB_1$  and  $LB_2$  stems from the fact that  $\alpha_i$  and  $\beta_i$  are valid lower bounds on the time span that is required for processing all the jobs in stage  $i$ .

Actually, it is possible to improve  $LB_2$  by noting that  $\gamma_i = \max_{i \in J} p_{ij}$  is also a valid lower bound on the time span that is required for processing all the jobs in stage  $i$ . Finally, a valid lower bound, that dominates all the other ones, is given by:

$$LB'_2 = \max_{i \in M} \left\{ \min_{j \in J} r_{ij} + \max\{\alpha_i, \beta_i, \gamma_i\} + \min_{j \in J} q_{ij} \right\} \quad (3)$$

**Remark:** All the bounds  $LB_1$ ,  $LB_2$ , and  $LB'_2$  can be computed in  $O(n)$ -time.

## 4.2 A new lower bound based on Dual Feasible Functions

Firstly, we recall that a function  $f$  is said to be discrete *dual feasible* if for any discrete finite set  $S$  of nonnegative integers, we have:

$$\sum_{x \in S} x \leq B \Rightarrow \sum_{x \in S} f(x) \leq f(B),$$

where  $B$  is a nonnegative integer.

The concept of Dual Feasible Functions (DFFs) has been introduced by Lueker [22] in the context of bin-packing. During the last decade, DFFs have been successfully used for deriving tight lower bounds for one-dimensional bin packing problems ([13], [15]) and two-dimensional bin packing problems as well [6]. We refer to [8] for an in-depth survey of DFFs.

In this section, we show how to use DFFs to derive enhanced lower bounds for the hybrid flow shop scheduling problem with multiprocessor tasks. To that aim, let  $I$  be an instance of the Hybrid Flow Shop scheduling problem with multiprocessor tasks with a corresponding optimal makespan  $C_{\max}(I)$ . Given  $m$  DFFs  $f_1, f_2, \dots, f_m$ , we associate to  $I$  a transformed instance  $\tilde{I}$  (with a corresponding optimal makespan  $C_{\max}(\tilde{I})$ ) that is obtained by substituting parameters  $size_{ij}$  by  $f_i(size_{ij})$  (for  $i \in M, j \in J$ ), and  $m_i$  by  $f_i(m_i)$  (for  $i \in M$ ).

**Proposition 1:** We have  $C_{\max}(\tilde{I}) \leq C_{\max}(I)$ .

*Proof:* It suffices to observe that if we consider any feasible schedule of instance  $I$  then we can derive a similar feasible schedule for instance  $\tilde{I}$  (with the same makespan). This result follows from the fact the  $f_i(\cdot)$ 's are DFFs and therefore the capacity constraints remain enforced after applying the transformation.  $\square$

An immediate consequence of this proposition is the following result.

**Corollary 1:** If  $L(\tilde{I})$  is a valid lower bound on  $C_{\max}(\tilde{I})$  then it is a valid lower bound on  $C_{\max}(I)$ .

We performed preliminary computational experiments and found evidence that a good performance is obtained through the combination of the two following DFFs. The first one,  $f_1^s$  ( $1 \leq s \leq m_i/2$ ), was proposed in [6]. It is defined as follows:

$$f_1^s : [0, m_i] \rightarrow [0, M(m_i, J_i)]$$

$$x \mapsto \begin{cases} M(m_i, J_i) - M(m_i - x, J_i) & \text{if } m_i > B/2 \\ x & \text{if } s \leq x \leq m_i/2 \\ 0 & \text{otherwise} \end{cases}$$

where  $M(\kappa, J_i)$  is the solution of the knapsack problem defined by items of the set  $J_i = \{j \in [1, n] : s \leq \text{size}_{ij} \leq \kappa/2\}$  ( $s = 1, \dots, \kappa/2$ ), capacity  $\kappa$ , and where the objective is to maximize the number of selected items.

The second DFF that we used  $f_2^\epsilon$  ( $\epsilon \in [0, \frac{1}{2}]$ ) has been proposed in Fekete and Schepers (1998) and is defined as follows:

$$f_2^\epsilon : [0, m_i] \mapsto [0, 1]$$

$$x \mapsto \begin{cases} 1 & \text{for } \frac{x}{m_i} > 1 - \epsilon \\ x & \text{for } \epsilon \leq \frac{x}{m_i} \leq 1 - \epsilon \\ 0 & \text{for } \frac{x}{m_i} < \epsilon \end{cases} \quad (4)$$

In so doing, and by varying parameters  $s$  ( $s \in [1, \frac{m_i}{2}]$ ) and  $\epsilon$  ( $\epsilon \in [0, \frac{1}{2}]$ ), we generate a set  $\Sigma$  of DFFs. In our implementation, for each instance  $I$ , we derive a set of 50 transformed instances that are obtained by selecting for each stage  $i$  a DFF  $f_i$  that is randomly drawn from  $\Sigma$ . For each transformed instance, we compute  $LB_3$ . Eventually, we keep as a final lower bound, the bound value that is computed over the 50 transformed instances. In the sequel, we shall refer to this DFF-based lower bound by  $LB^{DFF}$ .

## 5. Computational Study

### 5.1 Test Beds

To assess the performance of CDADS as well as the new proposed lower bound, we consider a set of 300 benchmark instances that is available on Ceyda Oğuz's home page ([http://home.ku.edu.tr/coguz/public\\_html/](http://home.ku.edu.tr/coguz/public_html/)). This benchmark is widely used in the literature ([17], [26], [31]). The number of jobs  $n$  is taken equal to 5, 10, 20, 50, 100 and the number of stages  $m$  is taken from the set  $\{2, 5, 8\}$ . The set of instances includes two types of problems. Each type being characterized by a specific machine distribution pattern. More specifically, for the instances of Type-1, the number of processors  $m_i$  available at each stage  $i$  is randomly drawn from the set  $\{1, \dots, 5\}$ , while for the instances of Type-2,  $m_i$  is set to 5 processors for every stage

*i.* For each combination of  $n$  and  $m$ , and for each type, 10 instances are randomly generated. The processing time of each job  $j$  in stage  $i$  ( $p_{ij}$ ) and its processing requirement ( $size_{ij}$ ) are integers and are randomly drawn from  $\{1, \dots, 100\}$  and  $\{1, \dots, m_i\}$ , respectively.

All the procedures were coded in C++ and run on an Intel Core 2 Duo 2 GHz PC. The time limit for CDADS was set equal to 60 seconds.

## 5.2 CDADS Evaluation

### 5.2.1 Restart Policy

A restart strategy has been implemented to take advantage of the fact that four priority rules are used to generate the initial solutions (see Section 3.2.1). For initialization, we use the most effective rule, *viz.* NSPT\_LS. However, if no improvement is obtained during the CDADS search, we restart the process with a new solution that is obtained by using the “Energy” rule and so on. The restart strategy is restricted by the size of the heuristics pool: restarts are then achieved at most four times, since we have selected four rules. The strategy parametrization follows the one given by Walsh in [33]: At the  $k^{th}$  restart (starting from  $k = 0$ ), the number of maximum nodes that can be visited is set to  $nbrNodes \times f^k$ , where  $f$  is empirically set to 1.3 and  $nbrNodes = 100 \times n$ . Hence, the search space is expanded at each restart.

### 5.2.2 Results

We tested two strategies for applying discrepancy: Top First and Bottom First. In the Top First exploration, discrepancies at the top of the tree are privileged while the Bottom First strategy favors discrepancies at the bottom. Computational study shows that CDADS is more effective with a Top First strategy (thus, contradicting – for the problem at hand – the statement of relative indifference of discrepancy claimed in [28]). Thus, the results shown below refer to this latter strategy. Table 3 gives for each configuration ( $n$ : number of jobs, and  $m$ : number of stages) and each type, the average percentage deviation ( $\%dev$ ) and the average CPU time. The average percentage deviation is measured in two ways:

1. For small problems, solutions are compared with the optimal solutions ( $C_{\max}^*$  denotes the optimum makespan):

$$\%dev = 100 \times \frac{C_{\max} - C_{\max}^*}{C_{\max}^*}$$

2. For larger problems, solutions found by the CDADS are compared with

the best computed lower bound ( $LB$ ):

$$\%dev = 100 \times \frac{C_{\max} - LB}{LB}$$

Since the  $FHm, ((PM^{(k)})_{k=1}^m) |size_{ij}| C_{\max}$  problem and the inverse problem (that is, the problem that is obtained by starting the processing route from the last stage and finishing it to the the first stage) have the same optimal makespan [34], then we consider a two-directional planning (forward schedule and backward schedule). We observe from Table 3, that the average percentage deviation is smaller for Type-1 instances (1.22 % for Type-1 problems vs 3.28 % for ‘Type-2’ problems).

Table 3: Performance of CDADS

$n$	$m$	‘Type-1’ Problems		‘Type-2’ Problems	
		$\%dev$	$CPU(s)$	$\%dev$	$CPU(s)$
5	2	0.00	< 0.1	0.00	< 0.1
	5	0.21	< 0.1	0.46	< 0.1
	8	1.31	< 0.1	0.50	< 0.1
10	2	0.00	< 0.1	0.84	< 0.1
	5	0.66	0.4	3.97	< 0.1
	8	5.51	< 0.1	7.32	0.2
20	2	0.41	0.1	0.30	3.1
	5	1.01	1.1	5.90	1.3
	8	3.67	0.2	10.37	1.3
50	2	0.20	2.3	0.26	4.2
	5	0.47	5.0	3.92	13.5
	8	1.47	6.8	4.99	33.4
100	2	0.07	11.1	2.67	22.8
	5	1.46	13.6	1.86	40.9
	8	1.85	11.0	5.85	47.3
<i>Global average</i>		1.22	3.44	3.28	10.53

Interestingly, we see that for a given  $n$ , the average percentage deviation increases as  $m$  increases.  $m$ . By contrast, for a given number of stages  $m$ ,

increasing  $n$  has no significant effect on the average percentage deviation. Regarding CDADS efficiency, it can be observed that this procedure converges quickly. Indeed, the average CPU time varies from less than 0.1 seconds to a maximum of 47.3 seconds for the large 100-job and 8-stage instances.

### ***5.2.3 Comparison of CDADS Solutions with State-of-the-Art Heuristics***

We compared the performance of CDADS with the following approaches: the genetic algorithm (GA) of [17], the constraint programming algorithm (CP) and the memetic algorithm (MA) of [17] (note that the results presented in [18] are not mentioned since they are less good than those presented in [17]). These three approaches are the most effective published so far. The results are displayed in Table 4. In this table, each entry (except for the last row) represents an average percentage deviation. Furthermore, the average CPU times are displayed in the last row of the table.

We computed the average percentage deviations with respect to the best derived lower bounds. In this regard, it is worth mentioning that we found evidence that the results published by Ercan *et al.* [24] include several inconsistencies due to miscalculations.

Table 4: Comparing average percentage deviation (and CPU time)

$n$	$m$	‘Type-1’ Problems				‘Type-2’ Problems			
		<i>CDADS</i>	<i>GA</i>	<i>CP</i>	<i>MA</i>	<i>CDADS</i>	<i>GA</i>	<i>CP</i>	<i>MA</i>
5	2	0.00	0.29	0.00	0.00	0.00	1.23	0.00	0.00
	5	0.21	1.35	0.00	0.00	0.46	1.44	0.00	0.00
	8	1.31	4.15	0.00	0.00	0.50	2.38	0.00	0.00
10	2	0.00	0.00	0.00	0.00	0.84	2.83	1.72	1.75
	5	0.66	1.64	0.00	0.00	3.97	7.80	6.10	5.67
	8	5.51	9.38	10.32	8.02	7.32	10.87	8.37	8.80
20	2	0.41	0.44	2.59	0.66	0.30	3.70	6.72	3.43
	5	1.01	3.49	10.85	2.78	5.90	9.57	22.86	9.57
	8	3.67	5.69	17.98	5.32	10.37	17.26	28.52	16.02
50	2	0.20	0.63	2.79	0.49	0.62	2.76	6.54	2.21
	5	0.47	0.59	5.30	0.51	3.92	10.95	20.01	10.32
	8	1.47	2.17	14.42	1.71	4.99	15.89	30.06	17.25
100	2	0.07	0.15	1.96	0.07	2.67	3.05	5.68	2.70
	5	1.46	2.50	5.19	2.33	1.86	14.95	19.13	14.37
	8	1.85	1.99	9.47	2.15	5.85	20.06	23.15	17.83
<i>Global average</i>		<b>1.22</b>	2.27	5.39	1.60	<b>3.28</b>	7.28	11.92	8.32
<i>Mean CPU(s)</i>		<b>3.44</b>	879.93	320.3	326.01	<b>10.53</b>	879.08	423.09	511.27

From Table 4, we can make the following observations:

- CDADS consistently outperforms all the other approaches. Indeed, for all problem sizes (except, for the tiny 5-job instances) CDADS exhibits the smallest deviations on both problems types.
- MA and GA are the second best approaches for Type-1 and Type-2 instances, respectively.
- CDADS is the fastest approach. This observation is confirmed by considering Dongarra’s normalized coefficients [10] that reveal that our machine is approximately only 3.5 times faster than the machine used by Jouglet *et al.* while methods proposed in [17] are much slower.

To further assess the effectiveness of CDADS, we provide in Table 5 the number of improved best known solutions. It can be seen from this table, that CDADS remarkably delivered 75 new best known solutions among the 300 test problems.

$n$	5	10	20	50	100	<i>Total</i>
‘Type-1’ Problems	0	1	5	8	8	<b>22</b>
‘Type-2’ Problems	0	0	10	20	23	<b>53</b>

### 5.3 Lower Bound Evaluation

We tested the effectiveness of  $LB^{DFP}$  on the same set of benchmark instances.

In Table 6, we report the results of the percentage deviations that are exhibited by  $LB'_2$  and  $LB^{DFP}$  (defined in Section 4). Note that the percentage deviation of a lower bound  $LB$  is given by  $100 \times \frac{C_{max}^* - LB}{C_{max}^*}$  if the optimum makespan is known, otherwise  $100 \times \frac{UB - LB}{UB}$ , where  $UB$  is the value of the best known upper bound. Thus, the column  $\%gap\ LB^{DFP}$  reports the percentage deviation we obtained for each configuration when  $LB^{DFP}$  is used as lower bound. Similarly, the column  $\%gap\ LB'_2$  shows the percentage deviation when  $LB'_2$  is the considered lower bound.

Table 6: Lower bound performance

$n$	$m$	‘Type-1’ Problems		‘Type-2’ Problems	
		$\%gap$ $LB^{DFP}$	$\%gap$ $LB'_2$	$\%gap$ $LB^{DFP}$	$\%gap$ $LB'_2$
5	2	3.31	6.35	3.66	18.25
	5	5.55	13.63	3.68	18.56
	8	3.80	11.18	2.30	10.23
10	2	0.31	0.56	2.62	5.92
	5	3.18	5.44	7.98	10.06
	8	6.80	12.26	9.3	14.41
20	2	0.29	0.37	0.23	2.96
	5	0.94	2.39	5.41	7.13
	8	3.22	4.03	8.58	12.19
50	2	0.16	0.44	0.23	1.92
	5	0.33	0.61	3.58	7.88
	8	1.12	1.26	4.70	10.73
100	2	0.06	0.06	1.70	2.40
	5	1.35	1.38	1.77	9.65
	8	1.20	1.27	5.49	12.38
<i>Global %gap</i>		<b>2.10</b>	4.10	<b>4.08</b>	9.64

Table 6 demonstrates the good performance of the proposed lower bound. The *Global %gap* on Type-1 problems, that is the average of all *%gaps*, is 2.1 for  $LB^{DFP}$  while it reaches 4.1 for  $LB'_2$ . Indeed, despite its simplicity,  $LB^{DFP}$  outperforms  $LB'_2$  while being fast (actually the required CPU time is about 1 ms for large instances). Furthermore, we see that instances of Type-2 often exhibit larger deviations which is a clear indication that these instances are harder to solve. Also, we observe that for both problem sizes, the deviations are generally increasing with the number of stages and decreasing with the number of jobs.

Pushing our analysis a step further, we performed a pairwise comparison of  $LB^{DFP}$  and  $LB'_2$ . The results are displayed in Table 7. In this table, the

column *Equal* reports the number of instances for which  $LB^{DFF}$  is able to give the same performance as  $LB'_2$ . *Equal* is being to take a value in the set  $\{0, \dots, 10\}$  since for each configuration ( $\{m$  stage,  $n$  jobs $\}$ ) and in each type of problems (Type-1 or Type-2), we are testing 10 instances. Thus, *e.g.*,  $Equal = 7$  means that among the 10 used instances,  $LB^{DFF}$  is performing with the same performance as  $LB'_2$  over 7 instances. Under the  $LB^{DFF}$  column, we show the number of instances on which  $LB^{DFF}$  outperforms  $LB'_2$ . Conversely,  $LB'_2$  column reports the number of times  $LB'_2$  surpasses  $LB^{DFF}$ .

We see from the results of the table that  $LB^{DFF}$  strictly outperforms  $LB'_2$  on 12 instances (out of 150) of Type-1, and 93 instances (out of 150) of Type-2. On the other hand,  $LB'_2$  never improves the results. Therefore our proposition  $LB^{DFF}$  dominates  $LB'_2$ . These results provide further evidence of the good performance of the DFF-based lower bound.

Table 7: LB comparison

		‘Type-1’ Problems			‘Type-2’ Problems		
		Best bound			Best bound		
$n$	$m$	<i>Equal</i>	$LB^{DFF}$	$LB'_2$	<i>Equal</i>	$LB^{DFF}$	$LB'_2$
5	2	7	3	0	1	9	0
	5	8	2	0	6	4	0
	8	9	1	0	7	3	0
10	2	10	0	0	5	5	0
	5	9	1	0	9	1	0
	8	8	2	0	1	9	0
20	2	10	0	0	6	4	0
	5	10	0	0	4	6	0
	8	10	0	0	2	8	0
50	2	8	2	0	5	5	0
	5	10	0	0	0	10	0
	8	10	0	0	2	8	0
100	2	10	0	0	7	3	0
	5	10	0	0	0	10	0
	8	9	1	0	2	8	0
<i>Global</i>		138	<b>12</b>	0	57	<b>93</b>	0

Finally, we report in Table 8 the results of an experiment that aims at assessing the overall contribution of this paper. In this table, each entry in the column entitled “New proposed bounds” represents the average percentage deviation  $100 \times \frac{UB^{new} - LB^{new}}{LB^{new}}$  where  $UB^{new}$  represents the value of the CDADS approach and  $LB^{new} = LB^{DF}$ . On the other hand, each entry in the column entitled “Bounds from the literature” represents the average percentage deviation  $100 \times \frac{UB^{lit} - LB^{lit}}{LB^{lit}}$  where  $UB^{lit}$  represents the value of the best so far published solution and  $LB^{lit} = LB'_2$ .

Table 8: Comparison of average deviations

	New proposed bounds	Bounds from the literature
Type-1 Problems	1.22	1.60
Type-2 Problems	3.28	8.32

The results displayed in Table 8 provide strong evidence that the proposed lower and upper bounding procedures are very effective for both problem types, and outperform state-of-the-art bounding procedures.

## 6. Conclusion

In this paper, we investigated the hybrid flow shop problem with multiprocessor tasks. We proposed a new discrepancy search method (CDADS) that is based on adjacent discrepancies. Also, we proposed a lower bound that is based on the concept of dual feasible functions. Our computational experiments provide strong empirical evidence that CDADS consistently outperforms the best heuristic approaches from the literature. In particular, CDADS successfully improved the best known solution of 75 benchmark instances. Furthermore, our computational study demonstrates that the dual feasible-based lower bound is often tighter than the best lower bound from the literature.

As a topic for future research, we recommend the derivation of a (first) exact procedure for solving the  $FHm, ((PM^{(k)})_{k=1}^m) |size_{ij}| C_{max}$ . We expect that the new derived upper and lower bounds would prove useful to achieve this challenging goal but this would require further investigation. Furthermore, it would be worth including specific resource constraint propagation techniques, especially energetic reasoning, which has already proved its performance on parallel machine systems.

## References

- [1] J. C. Beck and L. Perron. Discrepancy-bounded depth first search. In *Proc. of the Second International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'00)*, pages 8–10, Paderborn, Germany, 2000.
- [2] A. Ben Hmida, M. Haouari, M.-J. Huguet, and P. Lopez. Solving two-stage hybrid flow shop using climbing depth-bounded discrepancy search. *Computers and Industrial Engineering*, 60(2):320–327, 2010.
- [3] A. Ben Hmida, M.-J. Huguet, P. Lopez, and M. Haouari. Climbing depth-bounded discrepancy search for solving hybrid flow shop scheduling problems. *European Journal of Industrial Engineering*, 1(2):223–243, 2007.
- [4] S. Bertel and J.-C. Billaut. A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research*, 159(3):651–662, 2004.
- [5] G. Brooks and C. White. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal of Industrial Engineering*, 16(1):34–40, 1965.
- [6] J. Carlier, F. Clautiaux, and A. Moukrim. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers & Operation Research*, 34(8):2223–2250, 2007.
- [7] J. Chen and C.-Y. Lee. General multiprocessor task scheduling. *Naval Research Logistics*, 46(1):57–74, 1999.
- [8] F. Clautiaux, C. Alves, and J. Valério de Carvalho. A survey of dual-feasible functions and superadditive functions. *Annals of Operations Research*, 179(1):317–342, 2010.
- [9] M. Dal Cin and E. Dilger. On the diagnostability of self-testing multimicroprocessor systems. *Microprocessing and Microprogramming*, 7(3):177–184, 1981.
- [10] J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, University of Tennessee, 2011.

- [11] M. Drozdowski. Scheduling parallel tasks – Algorithms and complexity. In Leung J. Y-T., editor, *Handbook of Scheduling*. Chapman & Hall/CRC, 2004.
- [12] M. F. Ercan and Y.-F. Fung. Real-time image interpretation on a multi-layer architecture. In *Proceedings of IEEE TENCON'99*, pages 1303–1306, 1999.
- [13] S. P. Fekete and J. Schepers. New classes of lower bounds for bin packing problems. In *IPCO*, pages 257–270, 1998.
- [14] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1–3):23–47, 2003.
- [15] M. Haouari and A. Gharbi. Fast lifting procedures for the bin packing problem. *Discrete Optimization*, 2(3):201–218, 2005.
- [16] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 1, pages 607–615, Montréal, Québec, Canada, August 1995.
- [17] A. Jouglet, C. Oğuz, and M. Sevaux. Hybrid flow-shop: A memetic algorithm using constraint-based scheduling for efficient search. *Journal of Mathematical Modelling and Algorithms*, 8(2):271–292, 2009.
- [18] C. Kahraman, O. Engin, İ. Kaya, and R. E. Öztürk. Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach. *Applied Soft Computing*, 10(4):1293–1300, 2010.
- [19] J. E. Jr. Kelley. The critical-path method: Resources planning and scheduling. In Thompson G. L. and Muth J. F., editors, *Industrial Scheduling*, pages 347–365. Prentice-Hall, Englewood Cliffs, 1963.
- [20] Z. Kiziltan, A. Lodi, M. Milano, and F. Parisini. CP-based local branching. In Bessière C., editor, *LNCS*, volume 4741, pages 847–855. Springer, 2007.
- [21] R. E. Korf. Improved limited discrepancy search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, volume 1, pages 286–291, Portland, OR, August 1996.

- [22] G. S. Lueker. Bin packing with items uniformly distributed over intervals [a,b]. In *24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, pages 289–297. IEEE Computer Society, 1983.
- [23] M. Milano and A. Roli. On the relation between complete and incomplete search: An informal discussion. In *Proc. of the Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'02)*, pages 237–250, Le Croisic, France, 2002.
- [24] C. Oğuz and M. F. Ercan. A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *Journal of Scheduling*, 8(4):323–351, 2005.
- [25] C. Oğuz, Y.-F. Fung, M. F. Ercan, and X.-T. Qi. Parallel genetic algorithm for a flow shop problem with multiprocessor tasks. In *International Conference on Computational Science*, pages 548–559, Berlin, Heidelberg, 2003. Springer-Verlag.
- [26] C. Oğuz, Y. Zinder, V. Ha Do, A. Janiak, and M. Lichtenstein. Hybrid flow shop scheduling problems with multiprocessor task systems. *European Journal of Operational Research*, 152(1):115–133, 2004.
- [27] F. Parisini and M. Milano. Improving cp-based local branching via sliced neighborhood search. In *Symposium On Applied Computing - ACM SAC*, Taiwan, 2011.
- [28] P. Prosser and C. Unsworth. LDS: Testing the hypothesis. Technical Report DCS TR-2008-273, Dept. of Computing Science, University of Glasgow, 2008.
- [29] I. Ribas, R. Leisten, and J. M. Framiñan. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & OR*, 37(8):1439–1454, 2010.
- [30] R. Ruiz and J. A. Vázquez Rodríguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18, 2010.
- [31] F. S. Şerifoğlu and G. Ulusoy. Multiprocessor task scheduling in multi-stage hybrid flow-shops: A genetic algorithm approach. *Journal of the Operational Research Society*, 55(5):504–512, 2004.

- [32] A. Sprecher, R. Kolisch, and A. Drexl. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1):94–102, 1995.
- [33] T. Walsh. Depth-bounded discrepancy search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, volume 2, pages 1388–1395, Nagoya, Japan, August 1997.
- [34] K-C. Ying and S-W. Lin. Multiprocessor task scheduling in multistage hybrid flow-shops: An ant colony system approach. *International Journal of Production Research*, 44(16):3161–3177, 2006.