Climbing Depth-Bounded Discrepancy Search for Solving Hybrid Flow Shop Problems

Abir Ben Hmida[1,2], Marie-José Huguet[1], Pierre Lopez[1], Mohamed Haouari[2]

[1]Université de Toulouse, LAAS-CNRS, 7 avenue du Colonel Roche, Toulouse, France
(abenhmid@laas.fr, huguet@laas.fr, lopez@laas.fr)
[2]Ecole Polytechnique de Tunisie, Unité ROI, La Marsa, Tunisia
(mohamed.haouari@ept.rnu.tn)

ABSTRACT

This paper investigates how to adapt some discrepancy-based search methods to solve Hybrid Flow Shop (HFS) problems in which each stage consists of several identical machines operating in parallel. The objective is to determine a schedule that minimizes the makespan. We present here an adaptation of the Depth-bounded Discrepancy Search (DDS) method to obtain near-optimal solutions with makespan of high quality. This adaptation for the HFS contains no redundancy for the search tree expansion. To improve the solutions of our HFS problem, we propose a local search method, called Climbing Depth-bounded Discrepancy Search (CDDS), which is a hybridization of two existing discrepancy-based methods: DDS and Climbing Discrepancy Search. CDDS introduces an intensification process around promising solutions. These methods are tested on benchmark problems. Results show that discrepancy methods give promising results and CDDS method gives the best solutions.

Keywords: Scheduling, Hybrid Flow Shop, Discrepancy Search Methods, CDDS, Lower bounds, Heuristics

Climbing Depth-Bounded Discrepancy Search for Solving Hybrid Flow Shop Problems

ABSTRACT

This paper investigates how to adapt some discrepancy-based search methods to solve Hybrid Flow Shop (HFS) problems in which each stage consists of several identical machines operating in parallel. The objective is to determine a schedule that minimizes the makespan. We present here an adaptation of the Depth-bounded Discrepancy Search (DDS) method to obtain near-optimal solutions with makespan of high quality. This adaptation for the HFS contains no redundancy for the search tree expansion. To improve the solutions of our HFS problem, we propose a local search method, called Climbing Depth-bounded Discrepancy Search (CDDS), which is a hybridization of two existing discrepancy-based methods: DDS and Climbing Discrepancy Search. CDDS introduces an intensification process around promising solutions. These methods are tested on benchmark problems. Results show that discrepancy methods give promising results and CDDS method gives the best solutions.

Keywords: Scheduling, Hybrid Flow Shop, Discrepancy Search Methods, CDDS, Lower bounds, Heuristics

1. INTRODUCTION

In this paper, we consider the hybrid flow-shop (HFS) scheduling problem which can be stated as follows. Consider a set J={J_1, J_2, …, J_N} of N jobs and a set E={1,2,…,L} of L stages, each job is to be processed in the L stages. Solving the HFS problem consists in assigning a specific machine to each operation of each job as well as sequencing all operations assigned to each machine. Machines used at each stage are identical and let $M^{(s)}$ be the number of machines in the stage s. Successive operations of a job have to be processed serially through the L stages. Job preemption and job splitting are not allowed. The objective is to find a schedule which minimizes the maximum completion time, or makespan, defined as the elapsed time from the start of the first operation of the first job at stage 1 to the completion of the last operation of the last job at stage L.

The HFS problem is NP-Hard even if it contains two stages and when there is, at least, more than one machine at a stage [7]. Using popular three-field notation (see for example [13]), this problem can be denoted by FL(P)$\|C_{\max}$. Detailed reviews of the applications and solution procedures of the HFS problems are provided in [8][14][17][21].

Most of the literature has considered the case of only two stages. In [17], authors presented a case study in a two-stage HFS with sequence-dependent setup times and dedicated machines. For more general cases (i.e., with more than two stages), some authors developed a Branch and Bound (B&B) method for optimizing makespan, which can be used to find optimal solutions of only small-sized problem instances [2]. Later, this procedure has been improved in [23]. In this latter study, several heuristics have been developed to compute an initial upper bound and a genetic algorithm improves the value of this upper bound during the search. In order to reduce the search tree, new branching rules are proposed in [25]. Another B&B procedure for this problem is proposed by Carlier and Néron in [4]. They proved that their algorithm is more efficient than previous exact solution procedures.

Different heuristic methods were developed to solve HFS problems. Brah and Loo [3] expanded five standard flow shop heuristics to the HFS case and evaluated them with respect to Santos et al.'s lower bounds [24]. Recently, a new heuristic method based on Artificial Immune System (AIS) has been proposed to solve HFS problems [5] and proves its efficiency. Results of AIS algorithm have been compared with Carlier and Néron's lower bounds.

Lower bounds are developed in the literature which can be used to measure the quality of heuristic solutions when the optimal solution is unknown. Various techniques were proposed for obtaining lower bounds. In [16], authors reduce the HFS problem to the classical one and the optimal makespan of the latter one is a lower bound on the optimal makespan of the original problem. In [18], authors defined lower bounds based on the single-stage subproblem relaxation. The aggregation of the work yields a very rich class of lower bounds based on computing the total amount of work on some

stages or machines [6]. Brah and Hunsucker proposed two bounds for the HFS problem, one based on machines and another based on jobs [7]. Their lower bounds have been improved, later, by Portmann in [23].

The remainder of the paper is organized as follows. Section 2 gives an overview of discrepancy-based search methods. Section 3 presents how to adapt some of these methods to solve the HFS problem and details the lower bounds used. Section 4 is dedicated to an illustrative example to explain the proposed search methods. In Section 5, evaluations of the proposed methods on usual benchmarks are detailed. Finally we report some conclusions and open issues to this work.

2. DISCREPANCY-BASED SEARCH METHODS

Discrepancy-based methods are tree search methods developed for solving combinatorial problems. These methods consider a branching scheme based on the concept of discrepancy to expand the search tree. This can be viewed as an alternative to the branching scheme used in a Chronological Backtracking method.

The primal method, Limited Discrepancy Search (LDS), is instantiated to generate several variants, among them, Depth-bounded Discrepancy Search (DDS) and Climbing Discrepancy Search (CDS).

2.1 Limited Discrepancy Search

The objective of LDS proposed by Harvey in [12] is to provide a tree search method for supervising the application of some instantiation heuristics (variable and value ordering). It starts from an initial variable instantiation suggested by a given heuristic and successively explores branches with increasing discrepancies from it, i.e. by changing the instantiation of some variables. This number of changes corresponds to the number of discrepancies from the initial instantiation. The method stops when a solution is found (if such a solution does exist) or when an inconsistency is detected (the tree is entirely expanded).

The concept of discrepancy was first introduced for binary variables. In this case, exploring the branch corresponding to the best Boolean value (according a value ordering) involves no discrepancy while exploring the remaining branch implies one discrepancy. It was then adapted to suit to non-binary variables in two ways. The first one considers that choosing the first ranked value (rank 1) leads to 0 discrepancy while choosing all other ranked values implies 1 discrepancy. In the second way, choosing value with rank r implies r–1 discrepancies.

Dealing with a problem defined over N binary variables, an LDS strategy can be described as shown in Algorithm 1.

```
k ← 0    -- k is the number of discrepancies
k_max ← N  -- N is the number of variables
S_ref ← Initial_solution()      -- S_ref is the reference solution
while No_Solution() and (k < k_max) do
    k ← k+1
    -- Generate leaves at discrepancy k from S_ref
    -- Stop when a solution is found
    S_ref' ← Compute_Leaves(S_ref, k)
    S_ref ← S_ref'
end while
```

Algorithm 1. Limited Discrepancy Search

In such a primal implementation, the main drawback of LDS is to be too redundant: during the search for solutions with k discrepancies, solutions with 0 to k–1 discrepancies are revisited. To avoid this, Improved LDS method (ILDS) was proposed in [15]. Another improvement of LDS consists in applying discrepancy first at the top of the tree to correct early mistakes in the instantiation heuristic; this yields the Depth-bounded Discrepancy Search method (DDS) proposed in [26]. In the DDS algorithm, the generation of leaves with k discrepancies is limited by a given depth.
All these methods (LDS, ILDS, DDS) lead to a feasible solution, if it exists, and are closely connected to an efficient instantiation heuristic. These methods can be improved by adding local constraint propagation such as Forward Checking [11]. After each instantiation, Forward Checking suppresses inconsistent values in the domain of not yet instantiated variables involved in a constraint with the assigned variable.

2.2 Climbing Discrepancy Search

CDS is a local search method which adapts the notion of discrepancy to find a good solution for combinatorial optimization problems [20]. It starts from an initial solution suggested by a given heuristic. Then nodes with discrepancy equal to one are explored first, then those at discrepancy equal to 2, and so on. When a leaf with an

improved value of the objective function is found, the reference solution is updated, the number of discrepancies is reset to 0, and the process for exploring the neighborhood is again restarted (see Algorithm 2).

```
k ← 0    -- k is the number of discrepancies
k_max ← N  -- N is the number of variables
S_ref ← Initial_Solution()      -- S_ref is the reference solution
while (k < k_max) do
    k ← k+1
    -- Generate leaves at discrepancy k from S_ref
    S_ref' ← Compute_Leaves(S_ref, k)
    if Better(S_ref', S_ref) then
          -- Update the current solution
          S_ref ← S_ref'
          k ← 0
    end if
end while
```

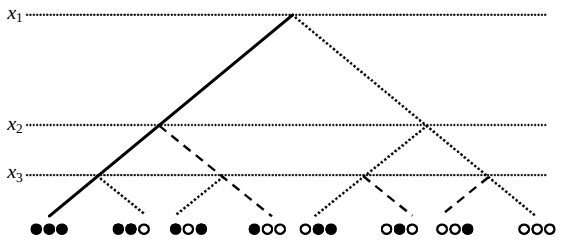
Algorithm 2. Climbing Discrepancy Search

The aim of CDS strategy is not to find only a feasible solution, but rather a high-quality solution in terms of criterion value. As mentioned by their authors, the CDS method is close to the Variable Neighborhood Search (VNS) [9]. VNS starts with an initial solution and iteratively explores neighborhoods more and more distant from this solution. The exploration of each neighborhood terminates by returning the best solution it contains. If this solution improves the current one it becomes the reference solution and the process is restarted. The interest of CDS is that the principle of discrepancy defines neighbourhoods as branches in a search tree. This leads to structure the local search method to restrict redundancies.

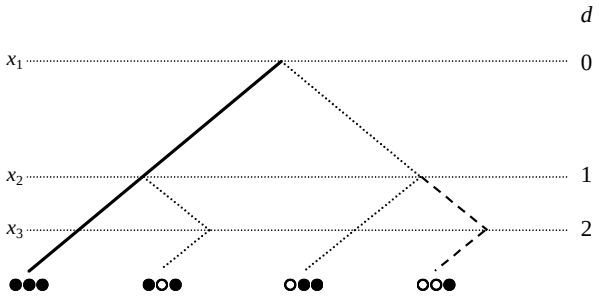
2.3. Example

As an example to illustrate the above exploration processes, let us consider a decision problem consisting of three binary variables x_1, x_2, x_3. The value ordering heuristic orders nodes left to right and, by convention, we consider that we descend the search tree to the left with $x_i = 0$, to the right with $x_i = 1$, $\forall$ i = 1,2,3. A solution is obtained with the instantiation of the three variables. Initially the reference solution S_{ref} is reached with the instantiation $[x_1\ x_2\ x_3] = [0\ 0\ 0]$. The solutions with 1 discrepancy from S_{ref} are those with one digit of $[x_1\ x_2\ x_3]$ equal to 1, e.g., [0 0 1]. To graphically represent the discrepancies that are performed to reach a solution (and also to be more homogeneous in the explanation of the different strategies), we associate a black circle to an instantiation which follows the value ordering heuristic whilst an open circle designates a discrepancy. In particular, following this semantics, S_{ref} is then associated to ●●● and the solution with one discrepancy on x_3 is associated to ●●○. Finally, the value of a given objective function f (suppose a minimization problem) is associated to a solution.

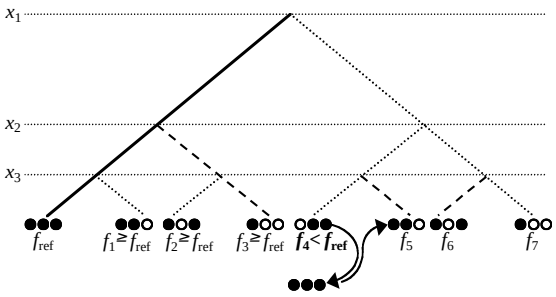
Figure 1 illustrates the search trees obtained using LDS (a), DDS (b), and CDS (c). For all these three methods, the search starts from a reference solution S_{ref} of value f_{ref} obtained with $[x_1\ x_2\ x_3] = [0\ 0\ 0]$. We see in Figure 1.a that the 8 leaves that are obtained with LDS correspond to the different solutions that are reachable from ●●●. In Figure 1.b, the tree contains 4 leaves only since the depth d is fixed at 2 and thus discrepancies can solely be made over x_1 and x_2. Figure 1.c illustrates the search tree obtained with CDS. The first reached solution from S_{ref} is of value f_1 with a corresponding discrepancy equal to 1. Since f_1 is greater than f_{ref}, then a second solution of value f_2 is generated. Again, its cost is compared with f_{ref} and this process is repeated until a solution of value f_4 having an improved cost is obtained. Thus, f_4 becomes the new reference solution. Therefore, the next solution (of criterion value f_5) is only at one discrepancy from this new reference solution.



a) LDS



b) DDS (d=2)



c) CDS

Figure 1. Three discrepancy search methods

3. DISCREPANCY-BASED METHODS TO SOLVE THE HYBRID FLOW SHOP PROBLEM

3.1 Problem Variables and Constraints

To solve the HFS problem under study, at each stage, we have to select a job, to allocate a resource for the operation of the selected job, and to fix its start time. Since the start time of each operation will be fixed as soon as possible to reduce the makespan, we only consider two kinds of variables: job selection and resource allocation. The values of these two kinds of variables are ordered following a given instantiation heuristic presented below.

At each stage *s*, we denote by X^s the job selection variables vector and by A^s the resource allocation variables vector. Thus, X_i^s corresponds to the i^{th} job in the sequence and A_i^s is its affectation value ($\forall i = 1,..., N$, with N the number of jobs). The domain of X_i^s variable is $\{J_1, J_2, \ldots, J_N\}$, $\forall i = 1,..., N$ and $\forall s = 1,..., L$ which corresponds to the choice of job to be scheduled. The values taken by the X_i^S variables have to be all different. The A_i^S domains are $\{1,\ldots, M^{(s)}\}, \forall i = 1,..., N$. Moreover, we consider precedence constraints between two consecutive operations of the same job and duration constraints for each operation at a given stage.

3.2 Discrepancy for Hybrid Flow Shop

Despite the fact we have two kinds of variables, we only consider here just one kind of discrepancy: *discrepancy on job selection variables*. Indeed, our goal is to improve the makespan of our solutions, and since all resources are identical, discrepancy on allocation variables cannot improve it. Thus, only the sequence of jobs to be scheduled may have an impact on the total completion time. More precisely, we aim at finding a good job order selection on the first stage. Next, stages 2,…,L are sequenced in turn. Each stage being sequenced using a specified priority rule. Hence a job selection order is defined for stage 1 and then a complete schedule is obtained through propagation. Clearly, an alternative strategy would require defining a specific job order selection for *each* stage. However, we have performed some preliminary computational experiments and we found that this latter strategy requires very long computer times without yielding significant better solutions.

Therefore, doing a discrepancy consists in selecting another job to be scheduled than the job given by a value ordering

heuristic. Job selection variables are *N*-ary variables. The number of discrepancy is computed as follows: the first value given by the heuristic corresponds to 0 discrepancy, all the other values correspond to 1 discrepancy (see Figure 2).

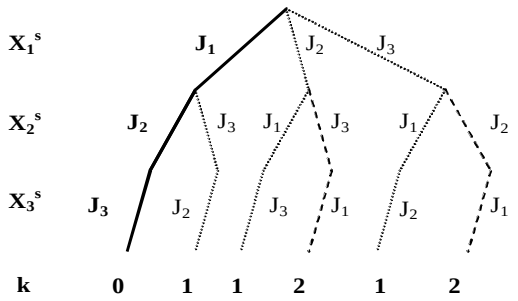


Figure 2. Discrepancies on job selection (stage s)

To obtain solutions of $k+1$ discrepancies directly from a solution with *k* discrepancies (without revisiting solutions with 0,…, *k-1* discrepancies), we consider the last instantiated variable having the k^{th} discrepancy value and we just have to choose a remaining variable for the $k+1^{th}$ discrepancy value.

At each stage *s*, the maximum number of discrepancy is $N-1$ which leads to develop a tree of $N!$ leaves (all the permutations of jobs are then obtained).

3.3 Instantiation Heuristics and Propagation

Variable ordering follows a *stage-by-stage* policy. The exploration strategy first consider job selection variable to choose a job, secondly consider resource allocation variable to assign the selected job to a resource.

We have two types of value ordering heuristics: the first one ranks jobs whilst the second one ranks resources.

Type 1: job selection. Several priority lists have been used. We first give the priority to the job with the earliest start time (*EST*) and in case of equality we consider three alternative rules: *SPT* (Smallest Processing Time) rule on the first stage, *LPT* (Longest Processing Time) rule on the first stage, and *CJ* (Critical Job) rule. The latter rule gives the priority to the job with the longest total duration.
We also consider all different combinations between these three heuristics. So, we give priority to the job having the earliest start time (*EST*) and, in case of equality, we consider *SPT* (respectively *LPT* / *CJ*) rule on the first stage and *LPT* or *CJ* (resp. *SPT* or *CJ* / *SPT* or *LPT*) in the second, and so on. The idea behind these combinations has the aim to mitigate the various configurations of machines.

Type 2: assignment of operations to machines. The operation of the job chosen by the heuristic of Type 1, is assigned to the machine such that the operation completes as soon as possible, that is, following an earliest completion time (*ECT*) rule. This latter rule is dynamic; the machine with the highest priority depends on the machines previously loaded.

After each instantiation of Type 2, we use a Forward Checking constraint propagation mechanism to update the finishing time of the selected operation and the starting time of the following operation in the job routing. We also maintain the availability date of the chosen resource.

3.4 Proposed Discrepancy-based Methods

In our problem, the initial leaf (with 0 discrepancy) is a solution since we do not constrain the makespan value. Nevertheless we may use discrepancy principles to expand the tree search for visiting the neighborhood of this initial solution. The only way to stop this exploration is to fix a limit for the CPU time or to reach a given lower bound on the makespan. To limit the search tree, one can use the *DDS method* which considers in priority variables at the top of the tree (job selection at the initial stages).

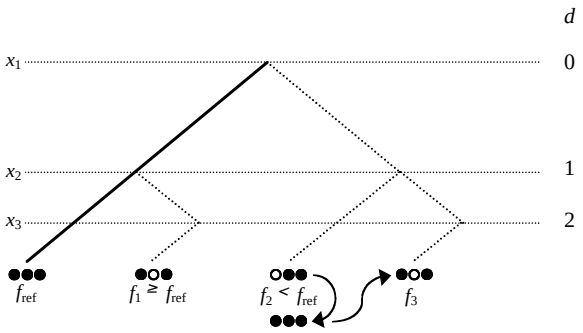
Thus, we first propose an adaptation of the initial DDS method based on the use of the variable ordering heuristics of types 1 & 2, joined with a computation of lower bounds at each node according to Portmann et al.'s rules [23] (see below).

Furthermore, to improve the search we may consider the CDS method which goes from an initial solution to a better one, and so on. The idea of applying discrepancies only at the top of the search tree can be also joined with the CDS algorithm to limit the tree search expansion. We have then developed a new strategy called CDDS method (Climbing Depth-bounded Discrepancy Search). With this new method, one can restrict neighborhoods to be visited by only using discrepancies on variables at the top of the tree (see Algorithm 3).

```
k ← 0    -- k is the number of discrepancy
k_max ← N  -- N is the number of variables
S_ref ← Initial_Solution()      -- S_ref is the reference solution
while (k < k_max) do
    k ← k+1
    -- Generate leaves at discrepancy k from S_ref
    -- and at d-depth value from the top of the tree with 1 ≤ d ≤ k
    S_ref'← Compute_Leaves(S_ref, k)
    if Better(S_ref', S_ref) then
          -- Update the current solution
          S_ref ← S_ref'
          k ← 0
    end if
end while
```

Algorithm 3. Climbing Depth-bounded Discrepancy Search

Figure 3 shows the tree obtained by CDDS from the examples depicted in Figures 1.b and 1.c



CDDS (*d*=2)

Figure 3. The Climbing Depth-bounded Discrepancy Search method

We can further enhance the CDDS strategy through the calculation of a lower bound at each node. So, we have the idea of introducing the lower bounds developed in [7] and improved in [23] which can be presented as follows (see also [14] as a survey presentation):

Suppose all jobs are sequenced on stages 1 through *L*-1 and a subset *Y* of jobs is already scheduled at stage *s*. Let consider $Sch^{(s)}(Y)$ a partial schedule of jobs Y at stage *s* and let $C[Sch^{(s)}(Y)]_m$ be the completion time of the partial sequence on machine m. Having fixed the schedule of jobs on the first *L*-1 stages and that of the jobs in Y at stage *s*, the average completion time of all jobs at stage *s*, $ACT[Sch^{(s)}(Y)]$, can be computed as follows:

$$ACT[Sch^{(s)}(Y)] = \frac{\sum_{m=1}^{M^{(s)}} C[Sch^{(s)}(Y)]_m}{M^{(s)}} + \frac{\sum_{j \in J-Y} p_j^{(s)}}{M^{(s)}} \qquad (1)$$

The expression of the maximum completion time of jobs in Y at stage *s*, $MCT[Sch^{(s)}(Y)]$, is given by

$$MCT[Sch^{(s)}(Y)] = \max_{1 \le m \le M^{(s)}} C[Sch^{(s)}(Y)]_m \qquad (2)$$

The machine based lower bound, LBM, is defined by

$$\begin{cases} ACT[Sch^{(s)}(Y)] + \min_{i \in J-Y} \left\{ \sum_{s'=s+1}^{L} p_i^{(s')} \right\} & \text{if} \quad ACT[Sch^{(s)}(Y)] \ge MCT[Sch^{(s)}(Y)] \end{cases}$$

$$\text{LBM}[\text{Sch}^{(s)}(Y)] = \begin{cases} \text{MCT}[\text{Sch}^{(s)}(Y)] + \min\limits_{i \in Y} \left\{ \sum\limits_{s'=s+1}^{L} p_i^{(s')} \right\} & \text{otherwise} \end{cases} \quad (3)$$

The job based lower bound, LBJ, is given by

$$\text{LBJ}[\text{Sch}^{(s)}(Y)] = \min_{1 \le m \le M^{(s)}} \{\text{C}[\text{Sch}^{(s)}(Y)]_m\} + \min_{i \in J-Y} \left\{ \sum_{s'=s}^{L} p_i^{(s')} \right\} \quad (4)$$

Finally, we obtain the composite lower bound, LBC, which given by

$$\text{LBC}[\text{Sch}^{(s)}(Y)] = \max\{ \text{LBM}[\text{Sch}^{(s)}(Y)], \text{LBJ}[\text{Sch}^{(s)}(Y)] \} \quad (5)$$

The LBM bound (3) is improved in Portmann et al. [23]. Namely, when $\text{ACT}[\text{Sch}^{(s)}(Y)] = \text{MCT}[\text{Sch}^{(s)}(Y)]$ and $J-Y=\emptyset$ then it may happen that

$$\min_{i \in Y} \left\{ \sum_{s'=s+1}^{L} p_i^{(s')} \right\} > \min_{i \in J-Y} \left\{ \sum_{s'=s+1}^{L} p_i^{(s')} \right\} \quad (6)$$

holds, for the processing times of the jobs in Y and J-Y are unrelated. In this case LBM can be improved by the difference of the left and right hand sides of (6). That is, when $J-Y=\emptyset$ the improved lower bound becomes

$$\text{LBM}[\text{Sch}^{(s)}(Y)] = \begin{cases} \text{ACT}[\text{Sch}^{(s)}(Y)] + \min\limits_{i \in J-Y} \left\{ \sum\limits_{s'=s+1}^{L} p_i^{(s')} \right\} & \text{if} \quad \text{ACT}[\text{Sch}^{(s)}(Y)] > \text{MCT}[\text{Sch}^{(s)}(Y)] \\ \text{MCT}[\text{Sch}^{(s)}(Y)] + \min\limits_{i \in Y} \left\{ \sum\limits_{s'=s+1}^{L} p_i^{(s')} \right\} & \text{if} \quad \text{ACT}[\text{Sch}^{(s)}(Y)] < \text{MCT}[\text{Sch}^{(s)}(Y)] \\ \text{ACT}[\text{Sch}^{(s)}(Y)] + \max\{ \min\limits_{i \in J-Y} \sum\limits_{s'=s+1}^{L} p_i^{(s')}, \min\limits_{i \in Y} \sum\limits_{s'=s+1}^{L} p_i^{(s')} \} & \text{if} \quad \text{ACT}[\text{Sch}^{(s)}(Y)] = \text{MCT}[\text{Sch}^{(s)}(Y)] \end{cases} \quad (7)$$

It is noteworthy that better bounds are available in the literature (see Haouari and Gharbi [10]). However, we have chosen to implement LBM for the sake of simplicity and efficiency.
In order to explain the global dynamics of the Climbing Depth-bounded Discrepancy Search method with the computation of lower bounds, Section 4 is dedicated to the description of an illustrative example. With this new method, one can restrict neighborhoods to be visited by using the evaluation of each visited node. This evaluation is ensured by the calculation of the lower bounds (see Algorithm 4).

```
k ← 0    -- k is the number of discrepancy
k_max ← N  -- N is the number of variables
S_ref ← Initial_Solution()      -- S_ref is the reference solution
UB ← C0  -- C0 is the value of the initial makespan
while (k ≤ k_max) do
       k ← k+1
       -- Generate leaves at discrepancy k from S_ref
       -- and at d-depth value from the top of the tree with 1 ≤ d ≤ k
       -- Each node such that LB(node) > UB is pruned
       S_ref' ← Compute_leaves (S_ref, k, UB)
       if Better(S_ref', S_ref) then
            -- Update the current solution
           S_ref ← S_ref'
           k ← 0
       end if
end while
```

Algorithm 4. Complete Climbing Depth-bounded Discrepancy Search (with lower bounds)

4. AN ILLUSTRATIVE EXAMPLE

Let consider a HFS of dimension 4×2 (i.e., 4 jobs and 2 stages) with the first stage composed of only one machine M_1, the second stage of two machines M_{21} and M_{22}. The allowed depth (d) is fixed at 2.

Table 1. Processing times of a 4×2 hybrid flow shop

Jobs	***Stage 1***		***Stage 2***	
1	O_{11}	8	O_{12}	7
2	O_{21}	7	O_{22}	8
3	O_{31}	8	O_{32}	8
4	O_{41}	7	O_{42}	8

Figure 4 shows the initial solution (0-discrepancy) provided by *EST-SPT* rule (in the first stage) which gives the following order for the job selection: (J_2,J_4,J_1,J_3) (the lexicographical order is applied for ties breaking). The makespan of the obtained solution is equal to 38. This latter value of makespan is considered as the first upper bound value of the problem: UB=38.

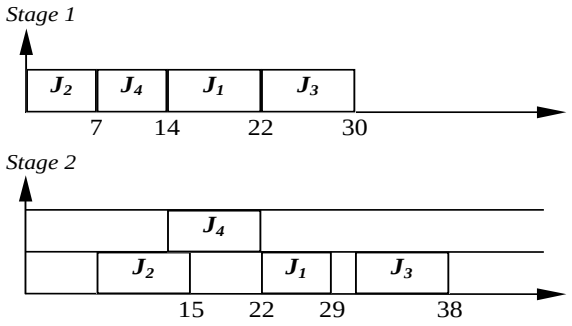


Figure 4. Initial solution

The neighbourhood associated to 1-discrepancy of this initial solution, as seen in Figure 5, consists of the following sequences (in bold the job upon which is done the discrepancy):

d=1	*d*=2
J4, J_2, J_1, J_3	J_2, ***J1***, J_4, J_3
J1, J_2, J_4, J_3	J_2, ***J3***, J_4, J_1
J3, J_2, J_4, J_1	

All resource allocation variables take the same value $A_i^1 = M_1$ because we have only one machine at the first stage.

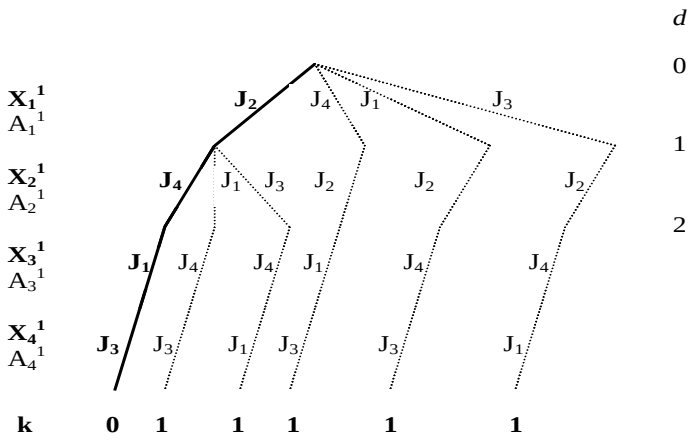


Figure 5. The neighborhood of the initial solution

For each of these sub-sequences, the next iteration of our algorithm schedules all the jobs and computes at each node the value of the lower bound. We use the LBC bound presented in Section 3. We start by the first sequence {J_4,J_2,J_1,J_3} and we calculate the LB at each node. Consider the subset Y = {J_4,J_2,J_1}, one has:

$$ACT[Sch^{(s)}(Y)] = \frac{22+8}{1} = 30 \; ; \; MCT[S^{(1)}(Y)] = 22.$$

Thus, LBM $[Sch^{(s)}(Y)] = 30 + 8 = 38$

and LBJ$[Sch^{(s)}(Y)] = 22 + 16 = 38$.

Consequently LBC$[Sch^{(s)}(Y)] = \max(38,38) = 38$.

This value of LB is equal to the initial upper bound. So, we prune this branch (see Figure 6). The same strategy is applied for the second sequence $\{J_1,J_2,J_4,J_3\}$ and we obtain at the first node LB=38. So, we stop the exploration of this branch.

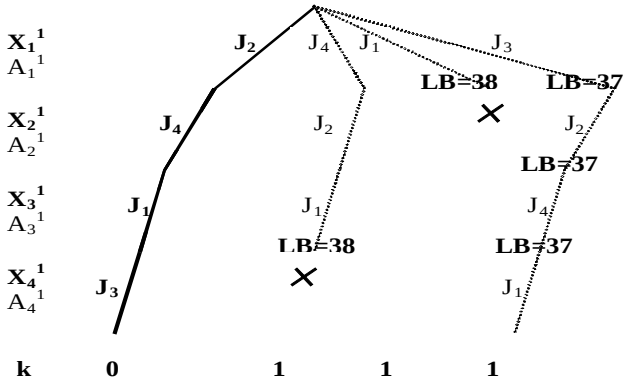


Figure 6. Neighbourhood's exploration

The third sequence$\{J_3,J_2,J_4,J_1\}$ gives a schedule of which the cost is C_{max} = 37 (Figure 7). This latter will be considered as the new reference solution and the number of discrepancy is reset to zero. So, we stop the exploration of the neighbourhood of the sequence $\{J_2,J_4,J_1,J_3\}$ and we define a new reference solution's neighbourhood. This later is shown in Figure 8. The upper bound is updated and it will be equal to 37 (UB=37).

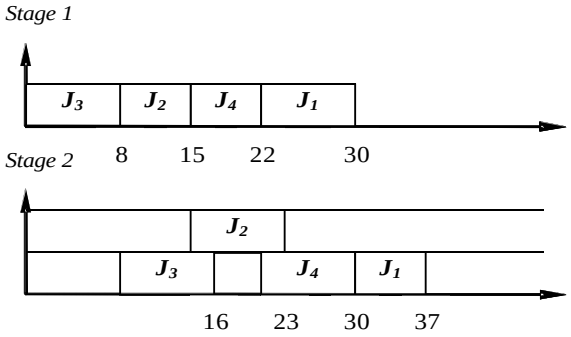


Figure 7. The third sequence solution

The neighbourhood associated to 1-discrepancy of the (new) reference solution is composed of the following sequences:

d=1	*d*=2
$\boldsymbol{J_2}$,J_3,J_4,J_1	J_3,$\boldsymbol{J_4}$,J_2,J_1
$\boldsymbol{J_4}$,J_3,J_2,J_1	J_3,$\boldsymbol{J_1}$,J_2,J_4
$\boldsymbol{J_1}$,J_3,J_2,J_4	

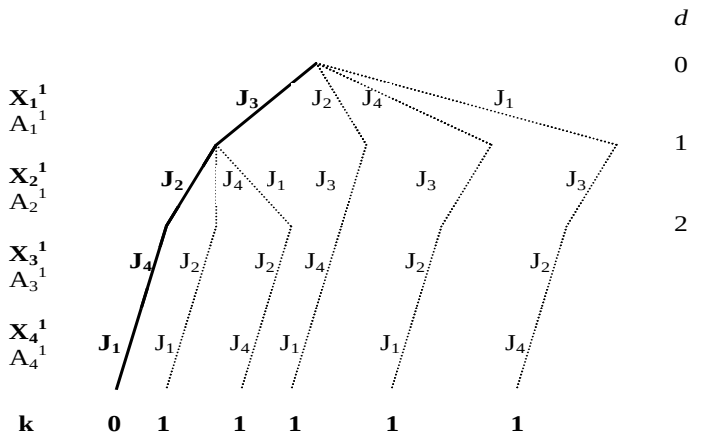


Figure 8. The neighbourhood of the new reference solution

The calculation of lower bounds at each node will guide the method for searching in promising branches (see Figure 9). The search will be stopped when there are no more branches to explore. The best solution is given in Figure 7 (C_{max}=37).

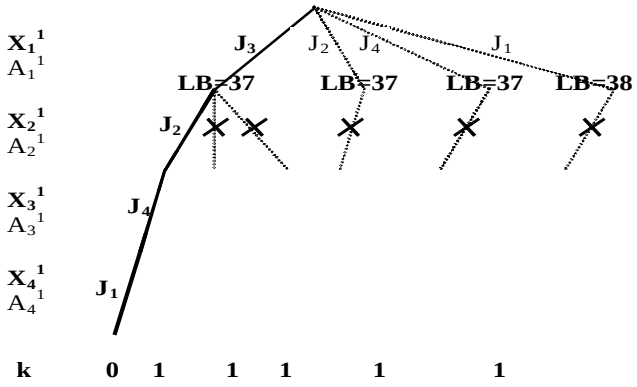


Figure 9. Neighbourhood's exploration for the new reference solution

5. COMPUTATIONAL EXPERIMENTS

5.1 Test beds

We compare our adaptation of the DDS method and our proposed CDDS method for solving a set of 77 benchmarks instances which are presented in [4][22]. In [22], all the problems have been solved using a Branch & Bound (B&B) method operating with use of satisfiability tests and time-bound adjustments. They calculated lower bounds (LBs) of the problems and they limited their search within 1800 s (thus, several instances were not solved to optimality). We also, compare DDS and CDDS methods with *AIS* strategy [5] which is, to the best of our knowledge, the most recent and best solution approach developed so far for solving the HFS.

In our study, we propose to compare our solutions with these LBs. We also run our algorithm within 1800 s. If no optimal solution was found within 1800 s, then the search is stopped and the best solution is output as the final schedule. The depth of discrepancy in our methods varies between 3 and 8 from the top of the tree. We have carried out our tests on a Pentium IV 3.20 GHz with 448 Mo RAM. DDS and CDDS algorithms have been programmed using C language and run under Windows XP Professional.

5.2. Results

In Table 2, for all considered problems, we present the best makespan values ($BestC_{max}$) obtained by DDS and CDDS methods among the different value ordering heuristics (*SPT, LPT, CJ, SPT-LPT, SPT-CJ, LPT-SPT, LPT-CJ, CJ-SPT,*

CJ-LPT), and the B&B algorithm of [22] within 1800 s. Deviation from LBs is calculated as follows:

$$\%\ deviation = \frac{BestC_{\max} - LowerBound}{LowerBound} \times 100$$

Lower bounds and deviations from such LBs are given in the last four columns.

In [22], some of the problems are grouped as hard problems. Hard problems consist of the c and d types of 10×5 and 15×5 problems where for configuration c, machines of the center are critical and there are two machines in the central stage, while in configuration d there are three machines at all stages. The rest of the problems (all a, b types, and 10×10 c type problems) are identified as easy problems. In configuration a, the machine of the central stage is critical and there is only one machine at this stage, while in configuration b the first stage is critical with only one machine. As shown in Table 2, for a and b type problems better results have been found than for c and d type problems. Indeed, the machine configurations have an important impact on problems complexity that affects solution quality [5].

Easy problems instances rapidly converge compared with hard ones. CDDS method takes 5 min in average to obtain all the solutions for easy problems, while DDS method takes 10 min in average. For hard problems, DDS algorithm takes 30 min and CDDS methods takes 25 min in average. Both of B&B and AIS algorithms take 4 min in average when resolving easy problems, while for hard problems B&B algorithm takes 25 min and AIS algorithm takes 10 min. Finally, note that both DDS and CDDS methods have been evaluated in the same computational environment while execution times of B&B and AIS are just reported from [22] and [5], respectively.

In Table 3, we compare the efficiency of the four methods for easy and hard problems. As it can be noticed from the table, for easy problems, DDS and CDDS algorithms provide better results than B&B, but for hard problems B&B algorithm and AIS strategy are better than DDS algorithm. Moreover, we observe that CDDS performs remarkably well on both problem classes since it yields better solutions than those provided by B&B and AIS methods.

Table 2. Solutions of test problems (bold problems have been identified as hard problems)

Problem	$C_{\max}$				*LB*	*% deviation (from LBs)*			
	B&B	*DDS*	*CDDS*	*AIS*		*B&B*	*DDS*	*CDDS*	*AIS*
J10c5a	111.6	111.6	111.6	111.6	111.6	0.0	0.0	0.0	0.0
J10c5b	122.7	122.7	122.7	122.7	122.7	0.0	0.0	0.0	0.0
J10c5c	**71.0**	**72.8**	**71.0**	**71.2**	**71.0**	**0.0**	**2.6**	**0.0**	**0.2**
J10c5d	**66.8**	**68.5**	**66.8**	**66.8**	**66.8**	**0.0**	**2.5**	**0.0**	**0.0**
J10c10a	149.8	150.8	149.8	149.8	149.8	0.0	0.7	0.0	0.0
J10c10b	163.0	163.2	163.0	163.0	163.0	0.0	0.1	0.0	0.0
J10c10c	128.0	118.8	116.5	117.0	108.0	19.1	10.5	6.7	8.8
J15c5a	161.8	162.3	161.8	161.8	161.8	0.0	0.4	0.0	0.0
J15c5b	161.2	161.5	161.2	161.2	161.2	0.0	0.2	0.0	0.0
J15c5c	**87 ;8**	**92.0**	**86.2**	**86.2**	**85.8**	**2.7**	**7.5**	**0.4**	**0.4**
J15c5d	**105.5**	**102.5**	**96.7**	**96.7**	**88.8**	**24.8**	**20.0**	**11.9**	**11.9**
J15c10a	207.0	207.5	206.8	206.8	206.8	0.1	0.3	0.0	0.0
J15c10b	211.8	211.8	211.8	211.8	211.8	0.0	0.0	0.0	0.0
Average						3.68	3.58	**1.62**	1.68

We found that both CDDS and AIS methods give optimal solutions for 61 instances out of 75. Moreover, for the remaining 14 instances, CDDS outperforms AIS for 3 instances, while AIS outperforms CDDS for only one instance.

Table 3. Relative efficiency of the four methods

Method	Easy problems	Hard problems
	% deviation	*% deviation*
B&B	2.21	6.88
AIS	1.01	3.12
DDS	1.42	8.01
CDDS	**0.96**	**3.06**

If all problems are considered, the average deviation from LBs for DDS algorithm is 3.58%, while the average deviation of B&B is 3.68% and for AIS is 1.679%. For CDDS the average is only of 1.627%.

Table 4 presents a comparison between the value ordering heuristics efficiency. For both DDS and CDDS methods, the third rule (*CJ*) gives always better solutions in a fixed running time.

Table 4. Efficiency of value ordering heuristics

heuristics	SPT	LPT	CJ	SPT-LPT	SPT-CJ	LPT-SPT	LPT-CJ	CJ-SPT	CJ-LPT
% deviation	*4.00*	*4.93*	*2.30*	*2.85*	*3.23*	*2.85*	*3.10*	*4.00*	*3.01*

Our discrepancy-based methods (DDS and CDDS) prove their contributions in terms of improvement of the initial makespan. Within 1800 seconds of CPU time, the deviation of the initial makespan has been reduced with DDS algorithm by nearly 14.7% for hard problems and 9.7% for easy ones. If we consider all problems, the initial makespan has been reduced with DDS algorithm by nearly 10.4%. For CDDS, the initial makespan reduction is about 12%. This percentage is distributed as 20.2% for hard problems and 8.25% for easy ones.

Enhancing CDDS with lower bounds computation has an important impact. Thus, CDDS method without integration of LBs has been developed and presented in a previous work [1] and its percentage deviation value from LBs was 2.32%.

6. CONCLUSION AND FUTURE RESEARCH

In this paper two discrepancy-based methods are presented to solve Hybrid Flow Shop problems with minimization of makespan. The first one is an adaptation of Depth-bounded Discrepancy Search (DDS) to suit to the problem under study. The second one, Climbing Depth-bounded Discrepancy Search (CDDS), combines both CDS and DDS. The two methods are based on instantiation heuristics which guide the exploration process towards some relevant decision points able to reduce the makespan. These methods include several interesting features, such as constraint propagation and lower bounds computations to prune the search tree, that significantly improve the efficiency of the basic approach. Computational results attest to the efficacy of the proposed approaches. In particular, CDDS outperformed the best existing methods.
Future work needs to be focused on improving the efficiency of the CDDS method. In particular, we expect that the use of the energetic reasoning [19] would significantly reduce the CPU time. But, this needs to be investigated thoroughly. Moreover, a second research avenue that requires investigation is the implementation of the CDDS method for other complex scheduling problems. In particular, we have already obtained promising results with CDDS for solving the flexible job shop problem.

REFERENCES

[1] Ben Hmida A., Huguet M.-J., Lopez P., Haouari M., “Adaptation of Discrepancy-based methods for solving Hybrid Flow Shop Problems”, *Proceedings IEEE-ICSSSM’06*, 1120-1125, 2006

[2] Brah S.A., Hunsucker J.L., “Branch and Bound algorithm for the flow shop with multiprocessors”, *European Journal of Operational Research* (51) 88-89, 1991

[3] Brah S.A., Loo L.L., “Heuristics for scheduling in a flow shop with multiple processors”, *European Journal of Operational Research* (113) 113-122, 1999

[4] Carlier J., Néron E., “An exact method for solving the multiprocessor flowshop”, *RAIRO-Operations Research* (34) 1-25, 2000

[5] Engin O., Döyen A., “A new approach to solve hybrid flow shop scheduling problems by artificial immune system”, *Future Generation Computer Systems* (20) 1083-1095, 2004

[6] Guinet A., Solomon M., Kedia P.K., Dussauchoy A., “A computational study of heuristics for two-stage flexible flowshops”, *International Journal of Production Research* (34) 1399-1415, 1996

[7] Gupta J.N.D., “Two-stage hybrid flowshop scheduling problem”, *Journal of the Operations Research Society* (39) 359-364, 1988

[8] Gupta J.N.D., “Hybrid flowshop scheduling problems”, *Production and Operational Management Society Annual Meeting*, 1992

[9] Hansen P., Mladenovic N., “Variable neighborhood search: Principles and applications”, *European Journal of Operational Research* (130) 449-467, 2001

[10] Haouari M., Gharbi A., “Lower bounds for scheduling on identical parallel machines with heads and tails”, *Annals of Operations Research* (129), 187-204, 2004
[11] Haralick R., Elliot G., “Increasing tree search efficiency for constraint satisfaction problems”, *Artificial Intelligence* (14) 263-313, 1980
[12] Harvey W.D., “Nonsystematic backtracking search”, *PhD thesis*, CIRL, University of Oregon, 1995
[13] Hoogeveen J.A., Lenstra J.K., Veltman B., “Preemptive scheduling in a two-stage multiprocessor flowshop is NP-Hard”, *European Journal of Operational Research* (89) 172-175, 1996
[14] Kis T., Pesch E., “A review of exact solution methods for the non-preemptive multiprocessor flowshop problem”, *European Journal of Operational Research* (164) 592-608, 2005
[15] Korf R.E., “Improved limited discrepancy search”, *Proceedings AAAI-96*, 286-291, 1996
[16] Lee C.Y., Vairaktarakis G.L., “Minimizing makespan in hybrid flow-shop”, *Operations Research Letters* (16), 149-158, 1994
[17] Lin H.T., Liao, C.J., “A Case Study in a Two-Stage Hybrid Flow Shop with Setup Time and Dedicated Machines”, *International Journal of Production Economics* (86) 133-143, 2003
[18] Linn R., Zhang W., “Hybrid flow shop scheduling: a survey”, *Computers & Industrial Engineering* (37) 57-61, 1999
[19] Lopez P., Esquirol P., “Consistency enforcing in scheduling: A general formulation based on energetic reasoning”, *Proceedings PMS’96*, 155-158, 1996
[20] Milano M., Roli A., “On the relation between complete and incomplete search: an informal discussion”, *Proceedings CPAIOR’02*, 237-250, 2002
[21] Moursli O., Pochet Y., “A branch and bound algorithm for the hybrid flow shop”, *International Journal of Production Economics* (64) 113-125, 2000
[22] Néron E., Baptiste P., Gupta J.N.D, “Solving an hybrid flow shop problem using energetic reasoning and global operations”, *Omega* (29) 501-511, 2001
[23] Portmann M.-C., Vignier A, Dardihac D., Dezalay D., “Branch and Bound crossed with G.A. to solve hybrid flow shops”, *International Journal of Production Economics* (43) 27-137, 1992
[24] Santos D.L., Hunsucker J.L., Deal D.E., “Global lower bounds for flow shops with multiple processors”, *European Journal of Operational Research* (80) 112-120, 1995
[25] Vignier A., *Contribution à la résolution des problèmes d’ordonnancement de type monogamme, multimachines (flow shop hybride)*, PhD Thesis, University of Tours, France, 1997 (in French)
[26] Walsh T., “Depth-bounded Discrepancy Search”, *Proceedings IJCAI-97*, 1388-1395, 1997