

Solving Two-Stage Hybrid Flow Shop Using Climbing Depth-bounded Discrepancy Search

Abir Ben Hmida^{1,2,3}, Mohamed Haouari³, Marie-José Huguet^{1,2}, Pierre Lopez^{1,2}

¹CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France

²Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France

(abenhmid@laas.fr, huguet@laas.fr, lopez@laas.fr)

³Ecole Polytechnique de Tunisie, Unité ROI, La Marsa, Tunisia
(mohamed.haouari@ept.rnu.tn)

Solving Two-Stage Hybrid Flow Shop Using Climbing Depth-bounded Discrepancy Search

ABSTRACT

This paper investigates how to adapt a discrepancy-based search method to solve two-stage hybrid flowshop scheduling 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 an adaptation of the Climbing Depth-bounded Discrepancy Search (CDDS) method based on Johnson's rule and on dedicated lower bounds for the two-stage hybrid flow shop problem. We report the results of extensive computational experiments, which show that the proposed adaptation of the CDDS method solves instances in restrained CPU time and with high quality of makespan.

Keywords: Scheduling, Two-stage Hybrid Flow Shop, Discrepancy Search Methods, CDDS, Lower Bounds, Heuristics.

1. INTRODUCTION

In this paper, we consider the two-stage Hybrid Flowshop Scheduling (HFS) problem which can be stated as follows. Consider a set $J=\{J_1, J_2, \dots, J_n\}$ of n jobs and two stages S_1 and S_2 , each stage S_i contains m_i identical machines ($i=1,2$). Successive operations of a job have to be processed serially through the two stages; each job $j \in J$ has to be processed first on a machine of stage S_1 during $p_{1,j}$ units of time. After that, job j has to be processed on a machine of the second stage S_2 for $p_{2,j}$ units of time. Job preemption and job splitting are not allowed. Moreover, a job cannot be processed by more than one machine at the same time and each machine processes at most one job at a time. Solving the two-stage HFS problem consists in assigning a specific machine to each operation of each job as well as sequencing all operations assigned to each machine. The objective is to find a schedule that minimizes the maximum completion time, or makespan, defined as the elapsed time from the start of the first operation of the first job in stage S_1 to the completion of the last operation of the last job in stage S_2 . Following the notation of [1], the considered problem is denoted $F2(P)\|C_{\max}$.

The $F2(P)\|C_{\max}$ problem is NP-Hard in the strong sense when there is, at least, more than one machine at a stage (*i.e.*, $\max(m_1, m_2) > 1$) [2]. Detailed reviews of the applications and solution procedures of the $F2(P)\|C_{\max}$ problems are provided in [3, 4, 5].

Most publications dealing with the two-stage hybrid flow shop problem assume that there is exactly one machine at one of the two stages [6, 2]. In the latter work, authors presented a case study in a two-stage HFS with sequence-dependent setup times and dedicated machines and they developed a branch-and-bound algorithm to solve it. For the general case (*i.e.*, with more than two machines in each stage), Lee and Vairaktarakis [7] show that the completion time obtained by applying the Johnson's rule on an auxiliary two-machine flow shop instance is a lower bound for the two-stage hybrid flow shop problem. The latter lower bound was improved in [8]. Authors show that their lower bound dominates the Lee and Vairaktarakis [7] one's. Guinet *et al.* [18] propose a heuristic for the makespan minimization problem in two-stage hybrid flow shop scheduling based on Johnson's rule. This heuristic is compared with the Shortest Processing Time (SPT) and the Longest Processing Time (LPT) dispatching rules. The authors conclude that the most effective approach used Johnson's rule to provide the priority list for job assignment. Haouari *et al.* [9] developed a branch-and-bound method to solve the problem and proved that their algorithm is more efficient than previous procedures. Tseng *et al.* [17] dealt with the issue of missing operations at the first stage and proposed some simple heuristics. The reader is referred to the survey paper of Ruiz and Vázquez-Rodríguez [16] for a recent overview on different methods proposed to solve general hybrid flow shop problems.

In this paper, we present some adaptations of a local search method, called *Climbing Depth-bounded Discrepancy Search* (CDDS), which has been initially proposed to solve HFS in the general case and has proved its efficiency to solve it [10]. To apply CDDS method to the particular case of $F2(P) \parallel C_{\max}$, we use the extended Johnson's rule developed in [7] to generate an efficient initial solution and we use the so-called SPT-based lower bound developed in [8] to prune the search tree. With these adaptations, the CDDS method has an excellent performance both in comparison to the general CDDS used to solve HFS and in comparison with other methods, and gives in the great majority of the cases a proven optimal solution. Furthermore, it provides high quality near-optimal solutions.

The remainder of the paper is organized as follows. Section 2 introduces the discrepancy-based search methods and details CDDS method. Section 3 presents how to adapt CDDS to solve the two-stage HFS problem and details the lower bounds used. Section 4 presents the results of a computational study of the proposed methods on well-known benchmark instances. Finally, we report some conclusions and open issues to this work.

2. CLIMBING DEPTH-BOUNDED DISCREPANCY SEARCH METHOD

Discrepancy-based methods are tree search methods developed for solving combinatorial constraint satisfaction problems. These methods are based on the depth-first search principle; however, when a dead-end occurs, the traditional chronological backtracking is replaced by the concept of discrepancy to expand the search tree. Harvey [11] has proposed the basic method called *Limited Discrepancy Search* (LDS). It consists in starting from an initial instantiation suggested by a given heuristic and successively explores branches by changing the instantiation of some variables: when considering one discrepancy, the instantiation of one variable is changed; with two discrepancies, the instantiation of two variables is changed, and so on. One can note that a given number of discrepancy leads to a set of instantiations. The search is stopped when a solution is found or when there is no solution (the entire tree is then expanded).

As an example to illustrate the above exploration processes, consider a decision problem consisting of four binary variables x_1, x_2, x_3, x_4 . 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 $x_i = 1$, $\forall i = 1,2,3,4$. A solution is obtained with the instantiation of the four variables.

Figure 1(a) illustrates the complete search tree obtained using LDS. At each node, a branch without discrepancy (in the left) is in plain line, while a branch corresponding to one discrepancy (in the right part of the tree) is drawn in dotted line. The first line below the leaves is associated with the discrepancy number, the second line with the order of apparition of each leaf during the search.

We consider the variant of LDS without redundancy: at the first iteration the leaf with a zero discrepancy is reached by the heuristic and during each other iterations, k leaves with exactly $k - 1$ discrepancies are reached. In the following, we use a variant of LDS in which discrepancies are applied first at the top of the tree to correct early mistakes of the instantiation heuristic. Thus in the tree of Figure 1(a), leaves with one discrepancy are obtained from the right to the left, as noted in the second line below the search tree.

To limit the tree search expansion, we also consider a *truncated variant* of the LDS method. Based on the idea introduced for the *Depth-bounded Discrepancy Search* method (DDS) [12], we limit the discrepancy to a given depth d . In this "truncated LDS", discrepancies are applied only on levels of which the depth is less or equal than d .

Figure 1(b) depicts the search tree obtained by the truncated LDS with $d=2$. In this tree, the leaves are reached in the same order as for the LDS method but there is no discrepancy at the level of depth 3; thus, the complete tree cannot be obtained.

These two methods (LDS and truncated LDS) are closely connected to an efficient instantiation heuristic.

Another variant of LDS dedicated to optimization problems is called *Climbing Discrepancy Search* (CDS) [13]. CDS can be viewed as a local search method that adapts the concept of discrepancy to find a good solution. It starts with an initial solution (zero discrepancy) and explores its neighborhood by the way of increasing discrepancies. If no solution with a better cost is found in this neighborhood, then the number of allowed discrepancies is increased. If a better solution is reached, the reference solution is updated, the number of discrepancies is reset to zero, and the process for exploring the neighborhoods is restarted. As mentioned by their authors, the CDS method is close to the Variable Neighborhood Search (VNS) [14]. The interest of CDS is that the principle of discrepancy defines neighborhoods as branches in a search tree. This leads to structure the local search method to restrict redundancies. The aim of CDS strategy is not to find only a feasible solution, but rather a high-quality solution in terms of criterion value.

For an overview on discrepancy-based methods the reader is referred to the survey paper by Milano and Roli [13].

Figure 2 illustrates the CDS method. In Figure 2(a), initially the reference solution S_{ref} is reached from the left branch of the tree, with zero discrepancy. By progressively increasing the number of discrepancies, CDS finds a better solution,

e.g., the seventh leaf with two discrepancies. In Figure 2(b), the method then restarts with this new reference solution associated with zero discrepancy, and explores the neighborhood of this solution to find a better solution (e.g., the fourth solution). One can note that there is no redundancy in a given iteration of the CDS method due to the LDS principle. However, some solutions can be reached several times between two iterations of the CDS method: for instance, the first solution in the second iteration of CDS was already obtained during the first iteration.

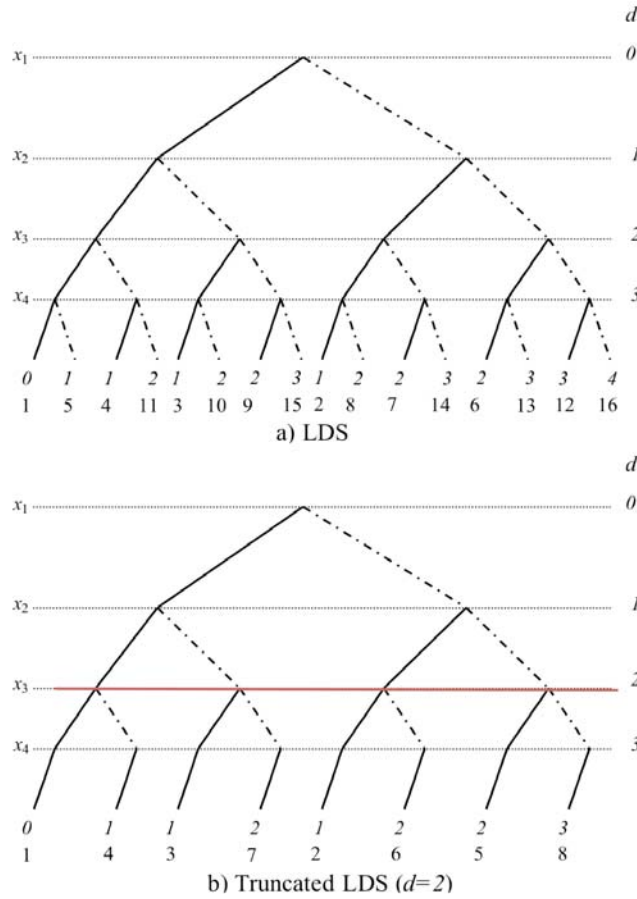


Figure 1. LDS and DDS search principles

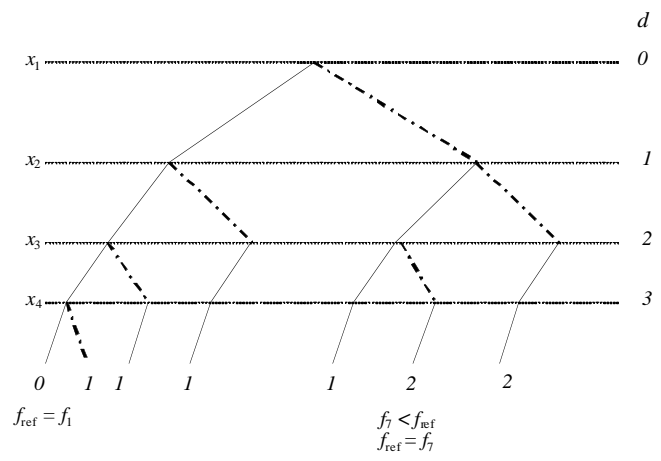
Joining the idea of truncated LDS (which limits the level at which discrepancies are done), and CDS (which updates the reference solution if a better solution is reached or increase the number of discrepancy otherwise), gives *CCDS* (Climbing Depth-bounded Discrepancy Search) [10]. With this method, one can restrict neighborhoods to be visited by only using discrepancies on variables at the top of the tree (see Algorithm 1). The `Compute_Leaves()` function generates leaves at discrepancy k from the reference solution and at d -depth value from the top of the tree.

```

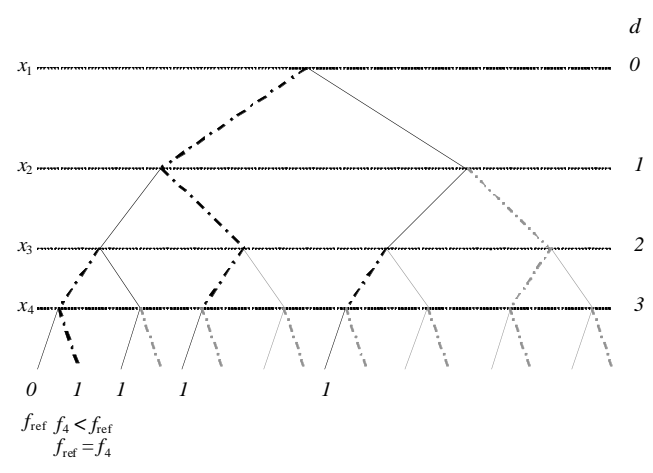
k ← 0    -- k is the number of discrepancy
k_max ← n -- n is the number of variables
S_ref ← Initial_Solution()    -- S_ref is the reference solution
while (k < k_max) do
  k ← k+1
  -- Generate leaves at discrepancy k from S_ref
  -- and at d-depth value from the top of the tree with 1 ≤ d ≤ k
  S_ref' ← 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 1. Climbing Depth-bounded Discrepancy Search



a) First iteration of CDS



b) Second iteration of CDS

Figure 2. CDS search principles

3. CDDS ADAPTATION TO SOLVE TWO-STAGE HYBRID FLOWSHOP SCHEDULING

To solve the two-stage HFS problem under study, we consider two kinds of variables: job selection and resource allocation stage by stage. The values of these two kinds of variables are ordered following a given instantiation heuristic presented below. The goal is to select an operation, to allocate a resource to the selected operation, and to set its start time. The start time of each operation will be set at the earliest possible value.

3.1 Discrepancy strategy

Since we wish to improve the makespan of our solutions, and since all resources are identical, discrepancy on allocation variables cannot improve it. As seen in Section 3.2, we use a list scheduling heuristic in which operations are assigned to the first available machine. There is then no freedom of resource allocation. Thus, we only consider the discrepancy on job selection variables. The reason is that only the sequence of jobs to be scheduled may have an impact on the makespan.

Therefore, achieving a discrepancy consists in selecting a next job to be scheduled rather than the job firstly suggested by the heuristic. Job selection variables are n -ary variables. The number of discrepancy is computed as follows: at each step, the first value given by the heuristic corresponds to 0 discrepancy, all the other values correspond to 1 discrepancy (see Figure 3). This binary counting mode provides diversified solutions.

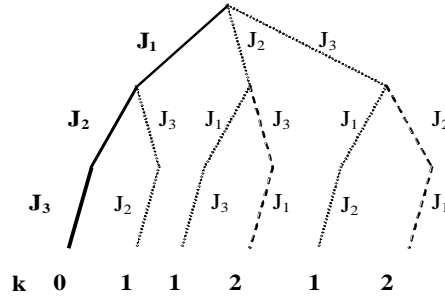


Figure 3. Discrepancies on job selection (stage s)

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

3.2 Heuristics

One could reasonably expect that the efficiency of the discrepancy-based methods depends closely on the quality of the initial solution [11]. Heuristics considered for the general Hybrid Flow Shop can be used [10] but also some dedicated heuristics for the two-stage HFS. As a dedicated heuristic, we schedule the operations at the first stage using the extension of the Johnson's rule to the two-stage HFS problem with at least two identical machines in each stage, proposed in [7]. In the second stage, we first give the priority to the operation belonging to the job having the earliest start time (EST) and in case of ties we consider the operation having the largest job duration.

Phase 1. Scheduling of Stage S_1

1.1. Order the jobs using the Johnson's rule. This rule is optimal if $m_1 = m_2 = 1$. It sequences a job i before a job j if $\min\{p_{1,i}; p_{2,j}\} \leq \min\{p_{1,j}; p_{2,i}\}$. In the considered problem we have $m_1 \geq 2$ and $m_2 \geq 2$, so we applied this rule for

the two-machine flow shop problem having $\left\{ \frac{p_{1,j}}{m_1}, \frac{p_{2,j}}{m_2}, j \in J \right\}$ as processing times [7]. Thus, we divide set J into two

disjoint subsets, J_1 and J_2 , where $J_1 = \left\{ j: \frac{p_{1,j}}{m_1} \leq \frac{p_{2,j}}{m_2} \right\}$ and $J_2 = \left\{ j: \frac{p_{1,j}}{m_1} > \frac{p_{2,j}}{m_2} \right\}$. Order the jobs in J_1 in the

non-decreasing order of $\frac{p_{1,j}}{m_1}$ and those jobs in J_2 in the non-increasing order of $\frac{p_{2,j}}{m_2}$. Sequence jobs in J_1 first,

followed by J_2 . Let $SEQ = J_1. J_2$.

1.2. Whenever a machine is idle, schedule a job $j \in SEQ$. Set $SEQ = SEQ \setminus \{j\}$,

1.3. If $SEQ \neq \emptyset$ then go to Step 1.2.

Phase 2. Scheduling of Stage S_2

2.1. For each job $j \in J$ set a release date $r_j = C_{1,j}$ (completion time of j on stage S_1). Set $SEQ = J$.

2.2. Whenever a machine is idle, schedule an already released job $j \in SEQ$ with earliest r_j and in case of ties with longest $p_{2,j}$. Set $SEQ = SEQ \setminus \{j\}$,

2.3. If $SEQ \neq \emptyset$ then go to Step 2.2. Else Stop.

Assignment of machines to operation is achieved by using the First Available Machine (FAM) rule.

After both instantiations, we use a simple calculation to update the finishing time of the selected operation as well as the starting time of the successor operation. We also maintain the availability date of the chosen resource.

3.3 Lower bounds

We can further enhance the CDDS strategy through the calculation of lower bounds on the makespan. Lower bounds dedicated to the general HFS can also be applied but we propose to introduce the SPT-rule based lower bound developed in [9] for the two-stage HFS, which can be presented as follows:

Let $I_2(J)$ be a lower bound on the total idle time in stage S_2 . This idle time is a direct consequence of the flow shop constraints. We take $I_2(J)$ equals to the minimum sum of completion times, on stage S_1 , of the m_2 jobs of J whose processing times are the shortest. Clearly, $I_2(J)$ can be obtained by applying the SPT rule. Thus,

$$LB_{SPT}^2(J) = \left\lceil \frac{I_2(J) + \sum_{j \in J} p_{2,j}}{m_2} \right\rceil$$

defines a lower bound. By using the symmetry of the hybrid flow shop problem (*i.e.*, by considering the inverse problem), we get the following lower bound:

$$LB_{SPT}^1(J) = \left\lceil \frac{I_1(J) + \sum_{j \in J} p_{1,j}}{m_1} \right\rceil$$

Hence, a valid lower bound is

$$LB_{SPT} = \max(LB_{SPT}^1, LB_{SPT}^2).$$

CDDS joined with the SPT-lower bound follows the scheme of the algorithm provided for implementing a general CDDS (Algorithm 1). The main difference is that the `Compute_Leaves()` function integrates an SPT-based lower bound computation at each node to prune the associated branch if this lower bound is greater than the current value of the makespan.

4. COMPUTATIONAL EXPERIMENTS

4.1 Test beds

Three sets of test problems have been considered. These instances were generated in a similar way as in [7].

- Set A: The number of jobs n was taken equal to 10, 20, 30, 40, 50, 100, and 150 jobs. The numbers of machines (m_1, m_2) are (2, 2), (2, 4), (4, 2), and (4, 4). The processing times are drawn randomly either from the discrete uniform distribution in [1, 20] for the first stage and in [1, 40] for the second stage. For each fixed n , there are 4 different combining problem characteristics. For each combination, 20 instances were generated. Hence, Set A contains a total number of 560 instances.
- Set B: This set contains 560 instances generated in the same way of Set A. However, the processing times on the first stage were drawn randomly from the discrete uniform distribution in [1, 40] and in [1, 20] for the second one.
- Set C: This set contains 560 instances generated in the same way of Set A. However, the processing times on both stages were drawn randomly from the discrete uniform distribution in [1, 40].

Thus, we considered a total of 1680 instances.

It is worth mentioning that the B&B algorithm of Haouari *et al.* optimally solved many of these instances [8]. However, this latter exact approach is significantly harder to implement than CDDS. Moreover, several small-sized instances remained unsolved by the exact method. Therefore, CDDS constitutes an appealing alternative solution strategy for deriving high quality solutions.

4.2 Performance analysis

In the first set of our experiments, we compare four variants of the CDDS method to evaluate the relative impact of both instantiation heuristics and lower bounds:

1. CDDS including heuristic and lower bound dedicated to the two-stage HFS, denoted by $CDDS^2$;
2. CDDS developed initially for solving the general hybrid flow shop problem [10] and including heuristic and lower bound for the general HFS with L stages, denoted by $CDDS^L$;
3. CDDS using the different heuristics mentioned in [10] for the HFS but joined with SPT-rule lower bounds dedicated to the two-stage HFS, denoted by $CDDS^{L-LB(2HFS)}$;
4. CDDS with the specific heuristic for the two-stage HFS and the lower bound for the general HFS, denoted by $CDDS^{2-LB(HFS)}$.

The proposed CDDS algorithms were coded in C and implemented on an Intel Core 2 Duo 2.9 GHz Personal Computer with 2GB of RAM. We set the maximum CPU time limit to 15 sec. If no optimal solution was found within 15 s, then the search is stopped and the best solution is output. Each instance and its inverse are successively considered. The depth of discrepancy is fixed at 7 from the top of the tree; this number has been experimentally selected.

The results of the computational study on Set A are summarized in Table 1. The column headings are as follows:

- n : number of jobs;
- (m_1, m_2) : numbers of machines in the first and second stage, respectively;
- US: number of instances that remain UnSolved (for which optimality was not proved after reaching the time limit);
- Time: average CPU time (in seconds) for all instances;
- Dev: Average deviation from the lower bound for all instances where the deviation is $\frac{BestC_{max} - LB_{SPT}}{LB_{SPT}} \times 100$ and

where $BestC_{max}$ is the best makespan obtained by CDDS method.

In Table 1, we observe that $CDDS^2$ outperforms $CDDS^L$, $CDDS^{L-LB(2HFS)}$, and $CDDS^{2-LB(HFS)}$ in terms of quality of solutions and CPU time. Indeed, it provides solutions of high quality within 84 seconds of total CPU time faced to 163 seconds for $CDDS^L$, 153 for $CDDS^{L-LB(2HFS)}$ and 121 for $CDDS^{2-LB(HFS)}$. On the other hand, we remark that the integration of the dedicated lower bound improves the quality of the produced solutions. Indeed, $CDDS^{L-LB(2HFS)}$ solved 70% of the test problems to optimality faced to 69% for $CDDS^L$, and 90% of problems were solved to optimality by $CDDS^2$ faced to 85% for $CDDS^{2-LB(HFS)}$. Furthermore, the results prove the efficiency of the dedicated heuristics for the two-stage HFS. Thus, when all problems are considered, the average deviation of $CDDS^2$ is strictly less than 0.19% faced to 0.91% for $CDDS^{L-LB(2HFS)}$ and of 0.36% for $CDDS^{2-LB(HFS)}$ faced to 0.96% for $CDDS^L$.

The global performance of the proposed method $CDDS^2$ is confirmed by the computational results that were obtained on Set B (see Table 2). Indeed, $CDDS^2$ provided an average optimality gap of 0.17% faced to 0.89% for $CDDS^L$, 0.85% for $CDDS^{L-LB(2HFS)}$, and 0.26% for $CDDS^{2-LB(HFS)}$. In this set, 91% of the test problems were optimally solved, faced to 73% for $CDDS^L$, 74% for $CDDS^{L-LB(2HFS)}$, and 87% for $CDDS^{2-LB(HFS)}$. Furthermore, considering the total CPU time, $CDDS^2$ is about twice times faster than $CDDS^L$ and $CDDS^{L-LB(2HFS)}$.

The experiments on Set C reported in Table 3 provide a further confirmation of the previous conclusions. Indeed, we observe that $CDDS^2$ provided solutions that deviate from the lower bounds by 0.26% while requiring 88 seconds of CPU time. On the other hand, $CDDS^L$ (resp. $CDDS^{L-LB(2HFS)}$ and $CDDS^{2-LB(HFS)}$) obtains solutions within 0.41% (resp. 0.39% and 0.30%) from LBs within 207 sec (resp. 191 sec. and 115 sec.). In this set, 86% of problems were solved to optimality by $CDDS^2$ faced to 55% for $CDDS^L$, 58% $CDDS^{L-LB(2HFS)}$ and 81% for $CDDS^{2-LB(HFS)}$.

General conclusions can be drawn from the analysis of Tables 1–3. First, the introduction of dedicated lower bounds had a marginal impact on the efficacy of the method. Indeed, we found that $CDDS^{L-LB(2HFS)}$ yielded 1% more optimal solutions than $CDDS^L$. This resulted in improving the average deviation by 0.03 and the average CPU time by 0.4 sec. This is confirmed comparing $CDDS^2$ and $CDDS^{2-LB(HFS)}$ (improvements of 5%, 0.10, and 0.88 sec., respectively). Secondly, dedicated heuristics are of great impact on these results since, going from $CDDS^L$ to $CDDS^{2-LB(HFS)}$, 18% more of problems are solved, the average deviation is improved by 0.44, and the CPU time by 2.25 sec. Moreover, going from $CDDS^{L-LB(2HFS)}$ to $CDDS^2$, 22% more of problems are solved, the average deviation is improved by 0.51, and the CPU time by 2.73 sec.

Table 1. Performance on Set A

n	(m_1, m_2)	$CDDS^2$			$CDDS^L$			$CDDS^{L-LB(2HFS)}$			$CDDS^{2-LB(HFS)}$		
		US	Dev	Time	US	Dev	Time	US	Dev	Time	US	Dev	Time
10	(2, 2)	3	0.21	3.62	11	1.20	8.79	11	1.20	8.28	5	0.37	5.97
	(2, 4)	10	2.47	8.28	17	6.24	12.83	15	5.42	11.31	13	3.20	9.81
	(4, 2)	0	0.00	0.07	3	0.06	2.31	3	0.06	2.26	2	0.03	1.82
	(4, 4)	0	0.00	0.13	4	0.12	3.10	4	0.12	3.03	1	0.02	1.61
20	(2, 2)	2	0.05	2.03	3	0.16	3.11	3	0.16	2.91	2	0.05	2.20
	(2, 4)	9	0.91	8.42	20	5.79	15.00	19	5.66	14.64	11	2.13	10.31
	(4, 2)	0	0.00	0.94	4	0.12	3.75	4	0.12	3.35	1	0.05	1.32
	(4, 4)	4	0.21	4.60	11	1.34	8.77	11	1.34	8.47	6	0.48	6.23
30	(2, 2)	1	0.02	0.92	7	1.61	5.85	6	1.39	4.75	3	0.11	3.23
	(2, 4)	6	0.86	6.89	17	5.61	12.92	16	5.54	12.88	9	2.05	11.51
	(4, 2)	0	0.00	0.45	3	0.12	2.63	3	0.12	2.25	1	0.07	1.80
	(4, 4)	3	0.07	3.78	8	0.32	6.76	8	0.32	6.56	5	0.16	5.09
40	(2, 2)	0	0.00	0.24	2	0.04	1.72	2	0.04	1.42	0	0.00	0.24
	(2, 4)	4	0.21	5.18	8	0.97	6.78	8	0.97	6.43	5	0.61	6.22
	(4, 2)	0	0.00	0.96	4	0.56	3.77	4	0.56	3.67	1	0.20	1.90
	(4, 4)	2	0.05	2.89	4	0.12	4.16	4	0.12	4.16	2	0.05	2.98
50	(2, 2)	0	0.00	0.40	3	0.07	2.59	3	0.07	2.39	1	0.03	0.96
	(2, 4)	2	0.15	2.37	6	0.70	5.33	6	0.70	5.23	3	0.23	3.56
	(4, 2)	0	0.00	0.60	4	0.11	3.48	4	0.11	3.18	1	0.04	2.35
	(4, 4)	2	0.06	3.91	4	0.08	4.56	4	0.08	4.53	4	0.08	4.55
100	(2, 2)	1	0.05	1.88	4	0.23	3.75	4	0.23	2.65	1	0.05	2.13
	(2, 4)	2	0.06	3.68	6	0.30	5.79	6	0.30	5.19	2	0.06	4.02
	(4, 2)	0	0.00	0.91	5	0.09	4.43	5	0.09	4.13	2	0.04	1.80
	(4, 4)	2	0.02	3.71	6	0.15	5.80	6	0.15	5.07	2	0.02	5.88
150	(2, 2)	0	0.00	4.40	3	0.30	5.99	3	0.30	5.12	1	0.07	4.99
	(2, 4)	0	0.00	2.41	2	0.16	3.67	2	0.16	3.66	1	0.10	2.98
	(4, 2)	0	0.00	5.27	3	0.18	6.73	3	0.18	6.13	0	0.00	6.01
	(4, 4)	1	0.01	8.10	2	0.05	8.79	2	0.05	8.34	1	0.01	8.21
Average		54	0.19	3.10	174	0.96	5.83	169	0.91	5.45	86	0.36	4.31
Total CPU time		84			163			153			121		

Table 2. Performance on Set B

n	(m_1, m_2)	$CDDS^2$			$CDDS^L$			$CDDS^{L-LB(2HFS)}$			$CDDS^{2-LB(HFS)}$		
		US	Dev	Time	US	Dev	Time	US	Dev	Time	US	Dev	Time
10	(2, 2)	2	0.09	1.51	5	0.12	4.32	5	0.12	3.32	3	0.10	2.27
	(2, 4)	0	0.00	0.44	5	0.38	4.08	5	0.38	3.98	2	0.16	1.71
	(4, 2)	9	1.60	7.03	19	5.92	14.28	18	5.72	13.82	12	2.33	9.71
	(4, 4)	2	0.29	1.52	7	1.18	5.74	7	1.18	5.34	4	0.53	3.32
20	(2, 2)	1	0.03	0.79	5	0.08	4.34	5	0.08	3.94	1	0.03	0.97
	(2, 4)	0	0.00	0.03	3	0.09	2.28	3	0.09	1.85	1	0.03	0.61
	(4, 2)	5	0.64	4.91	13	5.70	10.09	12	5.30	9.39	5	0.64	5.15
	(4, 4)	3	0.14	2.91	4	0.15	3.78	4	0.15	3.58	3	0.14	3.24
30	(2, 2)	0	0.00	0.03	4	0.16	3.02	4	0.16	3.02	0	0.00	0.34
	(2, 4)	0	0.00	0.10	3	0.12	2.34	2	0.09	2.04	1	0.07	1.23
	(4, 2)	7	0.93	6.05	11	5.69	8.64	11	5.69	8.64	8	1.13	6.78
	(4, 4)	2	0.11	2.49	3	0.19	3.31	3	0.19	3.11	3	0.23	2.77
40	(2, 2)	0	0.00	0.19	2	0.06	1.67	2	0.06	1.37	0	0.00	0.25
	(2, 4)	0	0.00	0.09	3	0.53	2.33	3	0.53	2.43	0	0.00	0.42
	(4, 2)	3	0.28	3.06	9	1.01	7.31	8	0.84	7.21	4	0.40	4.11
	(4, 4)	2	0.05	13.09	4	0.55	8.24	4	0.55	7.94	2	0.05	6.90
50	(2, 2)	0	0.00	0.65	3	0.56	2.80	3	0.56	2.60	1	0.11	1.33
	(2, 4)	0	0.00	0.10	2	0.43	1.59	2	0.43	1.49	1	0.07	1.51
	(4, 2)	8	0.37	7.23	10	0.69	7.95	9	0.47	7.45	9	0.47	7.44
	(4, 4)	1	0.02	2.94	3	0.06	4.75	3	0.06	4.35	2	0.13	3.32
100	(2, 2)	0	0.00	0.52	3	0.09	2.69	3	0.09	2.69	0	0.00	0.72
	(2, 4)	0	0.00	0.55	2	0.09	2.00	1	0.04	1.98	0	0.00	0.82
	(4, 2)	2	0.03	4.30	7	0.35	6.65	7	0.35	6.45	3	0.19	5.12
	(4, 4)	2	0.02	3.30	8	0.17	6.99	8	0.17	6.91	2	0.02	3.65
150	(2, 2)	0	0.00	1.39	3	0.08	3.43	3	0.08	3.41	1	0.04	2.88
	(2, 4)	0	0.00	0.90	4	0.13	3.72	3	0.10	3.63	1	0.03	1.67
	(4, 2)	1	0.03	3.03	5	0.22	6.02	5	0.22	6.01	1	0.03	3.43
	(4, 4)	1	0.01	5.06	3	0.03	6.55	3	0.03	6.43	2	0.34	5.34
Average		51	0.17	2.62	153	0.89	5.03	146	0.85	4.80	72	0.26	3.10
Total CPU time		73			141			134			87		

Table 3. Performance on Set C

n	(m_1, m_2)	$CDDS^2$			$CDDS^L$			$CDDS^{L-LB(2HFS)}$			$CDDS^{2-LB(HFS)}$		
		US	Dev	Time	US	Dev	Time	US	Dev	Time	US	Dev	Time
10	(2, 2)	3	0.20	2.35	9	0.26	7.18	9	0.26	6.84	5	0.24	3.45
	(2, 4)	1	0.34	0.81	6	0.45	5.07	6	0.45	4.73	3	0.39	0.92
	(4, 2)	0	0.00	0.06	5	0.05	3.80	3	0.03	3.10	0	0.00	0.36
	(4, 4)	9	1.89	6.75	18	1.98	13.58	16	1.95	13.22	12	1.91	9.45
20	(2, 2)	5	0.39	3.85	11	0.48	8.60	10	0.45	8.12	7	0.42	5.65
	(2, 4)	0	0.00	0.07	7	0.07	5.30	7	0.07	5.63	1	0.02	0.82
	(4, 2)	0	0.00	0.91	3	0.23	3.02	3	0.23	2.62	1	0.13	1.11
	(4, 4)	13	1.19	9.87	20	1.47	15.00	20	1.47	15.00	15	1.31	11.32
30	(2, 2)	4	0.10	3.07	12	0.63	9.31	12	0.63	9.71	7	0.24	5.45
	(2, 4)	0	0.00	0.44	8	0.08	6.26	7	0.05	5.76	0	0.00	0.97
	(4, 2)	1	0.02	1.18	5	0.05	4.64	5	0.05	3.24	1	0.02	1.35
	(4, 4)	9	1.42	9.10	14	1.95	10.80	14	1.95	10.67	11	1.76	9.85
40	(2, 2)	3	0.08	2.38	9	0.26	7.19	9	0.26	6.89	6	0.16	4.65
	(2, 4)	1	0.01	0.81	7	0.14	5.78	6	0.09	4.98	1	0.01	1.09
	(4, 2)	1	0.01	2.09	6	0.17	5.96	6	0.17	5.61	1	0.01	3.36
	(4, 4)	5	0.46	5.39	11	0.61	8.74	10	0.59	8.34	7	0.53	7.87
50	(2, 2)	0	0.00	0.19	6	0.06	4.63	6	0.06	4.31	0	0.00	0.34
	(2, 4)	0	0.00	0.26	8	0.28	6.16	8	0.28	6.01	0	0.00	0.62
	(4, 2)	2	0.02	1.88	5	0.05	4.46	5	0.05	4.31	3	0.03	2.34
	(4, 4)	5	0.84	4.89	12	1.20	9.39	11	1.16	8.90	8	0.94	6.38
100	(2, 2)	0	0.00	0.73	9	0.16	7.15	8	0.12	6.81	0	0.00	0.96
	(2, 4)	0	0.00	0.56	11	0.15	8.50	10	0.13	8.13	0	0.00	1.11
	(4, 2)	2	0.01	3.44	9	0.09	7.70	8	0.07	7.01	3	0.03	4.34
	(4, 4)	6	0.22	6.73	7	0.32	5.98	7	0.32	5.21	6	0.22	7.21
150	(2, 2)	1	0.01	4.28	8	0.08	8.57	7	0.06	8.01	1	0.01	5.13
	(2, 4)	0	0.00	1.62	9	0.09	7.64	9	0.09	7.12	0	0.00	1.82
	(4, 2)	0	0.00	5.04	7	0.07	8.53	6	0.05	7.91	0	0.00	6.14
	(4, 4)	7	0.11	9.69	10	0.14	8.19	10	0.14	7.94	9	0.12	11.02
Average		78	0.26	3.16	252	0.41	7.40	238	0.39	6.82	104	0.30	4.11
Total CPU time		88			207			191			115		

In the second part of experiments, we propose to compare solutions provided by $CDDS^2$ and results of the Tabu Search (TS) method developed in [9]. This latter method considered only 900 instances generated in a similar way as in [7]; the number of jobs n was restricted to 20, 30, 40, 50, and 100 jobs. The numbers of machines (m_1, m_2) considered are (2, 4), (4, 2), and (4, 4). Solutions of both methods are compared with SPT-lower bounds values calculated at the root (see Table 4).

As shown in Table 4, CDDS outperforms the Tabu Search method in all instances, except the instances of the subclass (4,2) of Set B. It is however assumed that the CDDS is very efficient for all problem sizes. In most problems, the CDDS procedure yields optimal or very near-optimal solutions. Considering all the problems, CDDS produced a proven *optimal* solution for 85.8% of the cases (772 out of 900) within 2.26 sec (considering the average CPU time) while TS yields the optimal solutions in 35% of the cases (316 out of 900) within 0.03 sec. Note that this latter value on TS CPU time was computed using the normalization coefficients of Dongarra [15].

Table 4. Performance comparison between the proposed $CDDS^2$ and TS [9]

n		Performance on Set A			Performance on Set B			Performance on Set C			Average
		(2, 4)	(4, 2)	(4, 4)	(2, 4)	(4, 2)	(4, 4)	(2, 4)	(4, 2)	(4, 4)	
20	$CDDS^2$	0.95	0.00	0.26	0.03	0.73	0.14	0.00	0.05	1.48	0.40
	TS	2.90	0.35	1.20	0.92	0.13	5.72	0.56	1.22	3.43	1.83
30	$CDDS^2$	0.92	0.00	0.10	0.00	0.96	0.11	0.07	0.02	1.45	0.40
	TS	1.43	0.06	0.85	0.57	0.05	3.10	0.27	1.46	1.45	1.03
40	$CDDS^2$	0.21	0.00	0.05	0.00	0.28	0.05	0.02	0.01	0.46	0.12
	TS	0.96	0.12	0.43	0.5	0.12	1.57	0.34	0.89	1.08	0.67
50	$CDDS^2$	0.15	0.00	0.06	0.00	0.37	0.02	0.00	0.02	0.88	0.16
	TS	0.54	0.02	0.30	0.26	0.04	1.09	0.20	0.42	0.95	0.42
100	$CDDS^2$	0.06	0.00	0.02	0.00	0.03	0.02	0.00	0.01	0.22	0.04
	TS	0.19	0.02	0.15	0.11	0.01	0.39	0.07	0.18	0.41	0.17
Average	$CDDS^2$	0.46	0.00	0.10	0.01	0.48	0.07	0.02	0.02	0.90	0.22
	TS	1.20	0.11	0.59	0.47	0.07	2.37	0.29	0.83	1.46	0.82

As displayed in Figure 4, we observe that the proposed $CDDS^2$ method performs better for larger problems. Considering all problems and all distributions, $CDDS^2$ presents a deviation from LBs of 0.22% versus 0.82% for TS method.

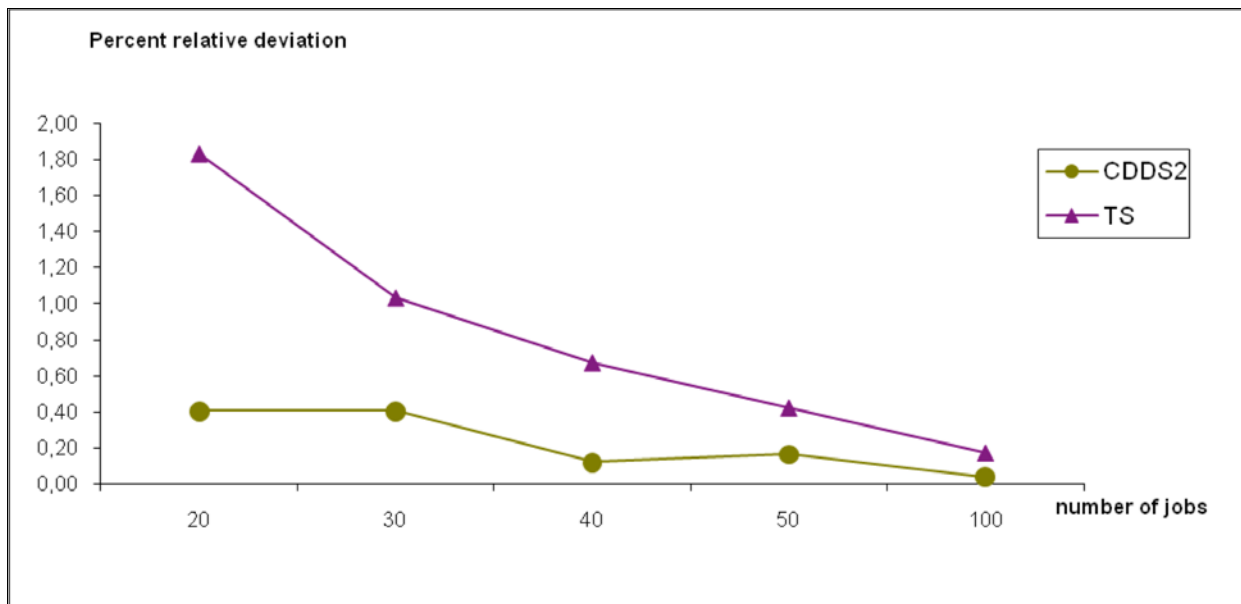


Figure 4. Relative deviation of $CDDS^2$ and TS for different problem sizes

5. CONCLUSIONS AND FUTURE RESEARCH

In this paper, we proposed an effective method to solve two-stage hybrid flow shop scheduling. This method is an adaptation of the Climbing Depth-bounded Discrepancy Search initially developed for the general hybrid flow shop problem. The adaptation integrates effective heuristics and lower bounds based on an extension of Johnson's rule. We presented extensive computational results which provide evidence that the proposed approach has an excellent performance, outperforms the algorithm developed for the general case, and consistently outperforms a well-known tabu search producing optimal solutions for the great majority of instances.

An important conclusion of the computational experience is that the key ingredient of a discrepancy-based search to solve two-stage flow shop scheduling relates to dedicated instantiation heuristics. In addition, the use of dedicated lower bounds improves slightly the efficiency of the method.

Future research efforts need to be focused on the adaptation of CDDS method for solving more realistic hybrid flow shop problems. In particular, as noticed by Ruiz and Vázquez-Rodríguez [16], a challenging research direction that deserves future investigation is concerned with the design of an effective CDDS-based solution approach to multiobjective HFS.

REFERENCES

- [1] Hoogeveen J.A., Lenstra J.K., Veltman B., “Preemptive scheduling in a two-stage multiprocessor flow shop is NP-Hard”. *European Journal of Operational Research*, 89:172–175, 1991.
- [2] Gupta J.N.D., Hariri A.M.A, Potts C.N., “Scheduling a two-stage hybrid flow shop with parallel machines at the first stage”. *Annals of Operations Research*, 69:171–191, 1997.
- [3] Narasimhan S.L., Panwalker S.S., “Scheduling in a two-stage manufacturing process”. *International Journal of Production Research*, 22:555–564, 1984.
- [4] Sherali H.D., Sarin S.C., Kodialam M.S., “Models and algorithms for a two-stage production process”. *Production Planning and Control*, 1:27–39, 1990.
- [5] Lin H.T., Liao C.J., “A case study in a two-stage hybrid flowshop with setup time and dedicated machines”. *International Journal of Production Economics*, 86:133–143, 2003.
- [6] Gupta J.N.D., “Two-stage hybrid flowshop scheduling problem”. *Journal of the Operational Research Society*, 39:359–364, 1988.
- [7] Lee C.Y., Vairaktarakis G.L., “Minimizing makespan in hybrid flow-shop”. *Operations Research Letters*, 16:149–158, 1994.
- [8] Haouari M., Hidri L., Gharbi A., “Optimal scheduling of a two hybrid flow shop”. *Mathematical Methods of Operations Research*, 64:107–124, 2006.
- [9] Haouari M., M’Hallah R., “Heuristic algorithms for the two-stage hybrid flowshop problem”. *Operations Research Letters*, 21:43–53, 1997.
- [10] Ben Hmida A., Huguet M.-J., Lopez P., Haouari M., “Climbing Discrepancy Search for solving the hybrid Flow shop”. *European Journal of Industrial Engineering*, 1(2):223–243, 2007.
- [11] Harvey W.D., “Nonsystematic backtracking search”. PhD thesis, CIRL, University of Oregon, 1995.
- [12] Walsh T., “Depth-bounded Discrepancy Search”. *Proceedings IJCAI-97*, p. 1388–1395, Nagoya, Japan, 1997.
- [13] Milano M., Roli A., “On the relation between complete and incomplete search: an informal discussion”. *Proceedings CPAIOR’02*, p. 237–250, Le Croisic, France, 2002.
- [14] Hansen P. and Mladenovic N., “Variable neighborhood search: principles and applications”. *European Journal of Operational Research*, 130:449–467, 2001.
- [15] Dongarra J., “Performance of various computers using standard linear equations software”. Computer Science Department, University of Tennessee, Knoxville, Tennessee, 1998.
- [16] Ruiz, R., Vázquez-Rodríguez, J.A., “The hybrid flow shop scheduling problem”. *European Journal of Operational Research*, 205(1):1–18, 2010.
- [17] Tseng, C. T., Liao, C. J., Liao, T. X., “A note on two-stage hybrid flowshop scheduling with missing operations”. *Computers & Industrial Engineering*, 54(3):695-704, 2008.
- [18] Guinet, A., Solomon, M.M., Kedia, P.K., Dussauchoy, A., “A computational study of heuristics for two-stage flexible flowshops”. *International Journal of Production Research*, 34(5):1399–1415, 1996.