Discrepancy Search for the Flexible Job Shop Scheduling Problem

Abir Ben Hmida, Mohamed Haouari, Marie-José Huguet, Pierre Lopez

Discrepancy Search for the Flexible Job Shop Scheduling Problem

Abir Ben Hmida[1,2,3], Mohamed Haouari[3], Marie-José Huguet[1,2], Pierre Lopez[1,2]

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

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

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

[3]Ecole Polytechnique de Tunisie, Unité ROI, La Marsa, Tunisia

(mohamed.haouari@ept.rnu.tn)

ABSTRACT

The Flexible Job Shop scheduling Problem (FJSP) is a generalization of the classical Job Shop Problem in which each operation must be processed on a given machine chosen among a finite subset of candidate machines. The aim is to find an allocation for each operation and to define the sequence of operations on each machine so that the resulting schedule has a minimal completion time. We propose a variant of the Climbing Discrepancy Search approach for solving this problem. We also present various neighborhood structures related to assignment and sequencing problems. We report the results of extensive computational experiments carried out on well-known benchmarks for flexible job shop scheduling. The results demonstrate that the proposed approach outperforms the best-known algorithms for the FJSP on some types of benchmarks and remains comparable with them on other ones.

Keywords: Scheduling, Allocation, Flexible Job Shop, Discrepancy Search, Neighborhood structures.

1. Introduction

The Flexible Job Shop Problem (FJSP) is a generalization of the traditional Job Shop scheduling Problem (JSP), in which it is desired to process operations on a machine chosen among a set of available ones. Therefore, the FJSP is more computationally difficult than the JSP, since it presents a further decision level besides the sequencing one, *i.e.*, the job assignment. Determining the job assignment requires assigning each operation on a specific machine chosen among a set of machines. This problem is known to be strongly NP-Hard even if each job has at most three operations and there are two machines [1].

The complexity of the FJSP suggests the adoption of heuristic methods producing reasonably good schedules in an acceptable execution time, instead of seeking for an exact solution. In recent years, the use of metaheuristics such as Tabu Search (TS) and Genetic Algorithms (GAs) has led to better results than classical dispatching or greedy heuristic algorithms [2, 3, 4].

In this paper, we present a new discrepancy-based method, called Climbing Depth-bound Discrepancy Search (CDDS), used initially for solving the hybrid flow shop problem [5]. Adapted to the problem under study, CDDS mixes, in an original way, some already known concepts and strategies. In particular, we use the block concept of Jurisch [6] to find the neighborhood structure. Computational experiments show that our algorithm outperforms the GA of Pezzella *et al.* [3] for all instances, and the TS of Mastrolilli and Gambardella [2] and the GA of Gao *et al.* [4] for some instances. Moreover, our algorithm proves that space exploration using the concept of discrepancy is suitable to develop efficient algorithms for the FJSP.

The remainder of the paper is organized as follows. In Section 2, we introduce the problem formulation and we provide a relevant-literature review. In Section 3, we present the considered method by detailing: the initial solution, the adapted discrepancy concept, the neighborhood structures, and the proposed algorithm. In Section 4, we present an extensive computational study comparing the results of different neighborhood structures and with the best-known approaches. Some conclusions are given in Section 5.

2. Problem formulation and previous works

The FJSP is formally formulated as follows. Let $J = \{J_i\}_{1 \le i \le N}$ be a set of N jobs to be scheduled where each job J_i consists of n_i operations. Let $O_{i,j}$ be the j^{th} operation of J_i. Let $M = \{M_k\}_{1 \le k \le m}$ be a set of m machines. Each machine can process only one operation at a time (disjunctive resource). Each operation $O_{i,j}$ can be processed without interruption on a unique machine R, selected among a given subset $M_{i,j} \subseteq M$; note that a given machine is able to process different operations of every job $J_i \in J$ (that is, re-circulation may occur). Given a schedule, we denote by $st_{i,j}$ and $C_{i,j}$ the starting time and completion time of operation $O_{i,j}$ ($1 \le i \le N$, $1 \le j \le n_i$), respectively. The objective is to find a schedule having a minimum completion time (or, *makespan*), denoted by $C_{\max} = \max_{i=1..N}(C_i)$, where $C_i = \max_{1 \le j \le n_i}(C_{i,j})$ is the completion time of job J_i.

Moreover, we shall assume that:

- All machines are available at time 0;
- All jobs are released at time 0;
- The order of operations for each job is predefined and cannot be modified.

Brucker and Schlie [7] propose a polynomial algorithm for solving the FJSP with two jobs, in which the processing times are identical whatever the machine chosen to perform an operation. Based on the observation that the FJSP turns into the classical job-shop scheduling problem when a routing is chosen, Brandimarte [8] was the first to use a decomposition approach for the FJSP. He solved the assignment problem using some dispatching rules and then solved the resulting job shop subproblems using a tabu search (TS) heuristic. Jurisch [6] considered the routing and scheduling decisions simultaneously and proposed a TS algorithm to solve it. Hurink *et al.* [9] and Barnes and Chambers [10] developed TS algorithms to solve the FJSP. Dauzère-Pérès and Paulli [11] defined a new neighborhood structure for the problem where there was no distinction between reassigning and resequencing an operation, and proposed a TS algorithm. Mastrolilli and Gambardella [2] improved the previous work and presented two neighborhood functions. Their TS is the well-known efficient approach for solving FJSPs. In [12], a genetic algorithm (GA) based on a discrete dynamic programming approach was proposed. In [13], Kacem *et al.* proposed a localization approach to solve the resource assignment problem, and an evolutionary approach controlled by the assignment model for the problem. Zhang and Gen [14] proposed a multistage operation-based GA to

deal with the problem based on a dynamic programming model. In [15], the authors treated this problem with a hybrid of particle swarm optimization and simulated annealing as a local search algorithm. Fattahi *et al.* [16] proposed a mathematical model and hybridizing of two methods, TS and simulated annealing. They developed six different algorithms for the FJSP. Pezzella *et al.* [3] presented a GA in which a mix of different strategies for generating the initial population, selecting individuals for reproduction, and reproducing new individuals is presented. Tay and Ho [17] proposed dispatching rules obtained by the genetic programming framework to solve the multiobjective FJSP with respect to minimum makespan, mean tardiness, and mean flow time objectives. Gao *et al.* [4] studied the FJSP with three objectives: minimization of the makespan, minimization of the maximum machine workload, and minimization of the total workload. They developed a hybrid genetic algorithm (hGA) based on the integrated approach for this problem. Their algorithm is the well-known competitive GA for solving the FJSP.

It is noteworthy that, to the best of our knowledge, the use of a discrepancy-based method had not been previously used for the flexible job shop problem.

3. Discrepancy-based search methods

Discrepancy-based methods are tree search methods developed for solving hard 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 basic method, Limited Discrepancy Search (LDS), is instantiated to generate several variants, among them, Depth-bounded Discrepancy Search (DDS) and Climbing Discrepancy Search (CDS).

3.1 Limited Discrepancy Search

The objective of LDS, proposed by Harvey [18], is to provide a tree search method for supervising the application of some ordering heuristics (variable and value ordering). It starts from initial variable instantiations suggested by a given heuristic and successively explores branches with increasing discrepancies from it, *i.e.*, by changing the instantiation of some variables. Basically, a discrepancy occurs if the choice of a variable does not follow the rank of the ordering heuristic (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 just one discrepancy. It was then adapted to suit to non-binary variables in two ways (Figure 1). The first one considers that choosing the first ranked value (rank 1, denoted by the instantiation of X by V_0 in the figure) 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.

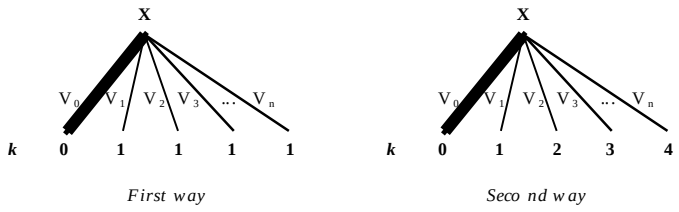


Figure 1. Counting discrepancies

We choose to adapt the first discrepancy definition to our work; this choice has been experimentally chosen (see [19] for more details). To more deeply understand the adopted discrepancy mechanism, see below and illustration in Figure 2.

Dealing with a problem defined over *N* binary variables, an LDS strategy can be described as shown in Algorithm 1. In this algorithm, **Initial_Instantiation()** exhaustively instantiates all variables following a given heuristic, while the function **No_Solution()** decides whether an instantiation has to be rejected as part of a solution (a solution corresponds to a feasible assignment for all the variables). **Compute_Leaves()** generates leaves at discrepancy *k* from the initial instantiation (by changing the instantiation of *k* variables), and stops when a solution is found and returns it.

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

Algorithm 1. Limited Discrepancy Search

In such a basic implementation, the main drawback of LDS is to be highly 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 [20]. 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 [21]. In the DDS algorithm, a given depth limits the generation of leaves with k discrepancies.

All these methods (LDS, ILDS, DDS) lead to a feasible solution, if it exists, and are closely connected to an efficient instantiation heuristic.

3.2 Climbing Discrepancy Search

Climbing Discrepancy Search (CDS) is a *local search* method that adapts the concept of discrepancy to find a good solution for combinatorial optimization problems [22]. It starts from an initial solution delivered by a given heuristic. Then, a sequence of neighborhoods including nodes with discrepancy equal to 1, 2,…, k_{max}, respectively, are consecutively explored (that is, we start with a neighborhood including nodes having a discrepancy equal to 1, then those 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 neighborhoods is restarted. A pseudo-code of CDS is displayed in Algorithm 2. In this algorithm, **Initial_Solution()** provides a feasible instantiation for all variables following a given heuristic and then returns the first reference solution. Moreover, **Compute_Leaves()** generates leaves at discrepancy k from it, and stops when an improved solution is reached or when its neighborhood is entirely explored.

```
k ← 0        -- k is the number of discrepancies
k_max ← N    -- N is the number of variables
Sol ← Initial_Solution()      -- Sol is the first reference solution
                              -- suggested by the heuristic
while (k < k_max) do
   k ← k+1
   do
     Sol' ← Compute_Leaves(Sol, k)  -- Generate leaves at discrepancy k
                                    -- from Sol
     If Better(Sol', Sol) then
        Sol ← Sol'    -- Update the current solution
        k ← 0
     end if
   until(Compute_Leaves(Sol, k) = ∅)
end while
```

Algorithm 2. Climbing Discrepancy Search

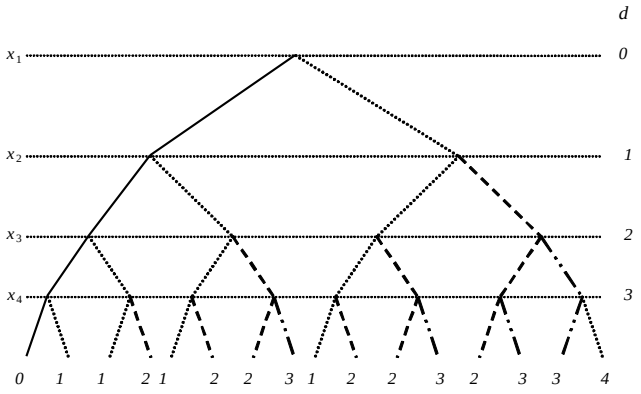
The aim of CDS strategy is to find a feasible solution of high-quality in terms of criterion value. As mentioned by their authors, the CDS method is close to Variable Neighborhood Search (VNS) [23]. Roughly speaking, both approaches explore the solution space using similar strategies. Indeed, VNS starts with an initial solution and iteratively explores neighborhoods that are 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.

It is important to stress the fact that by using a tree search to explore leaves in the same order as done by VNS, we can *remember*, in a sense, which leaves have been explored, contrarily to VNS method which moves from a solution to a better one, without storing information about the visited search space regions. Therefore it cannot know whether it is revisiting solutions or it has entirely explored the search space [22]. 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 by restricting redundancies following discrepancy principle.

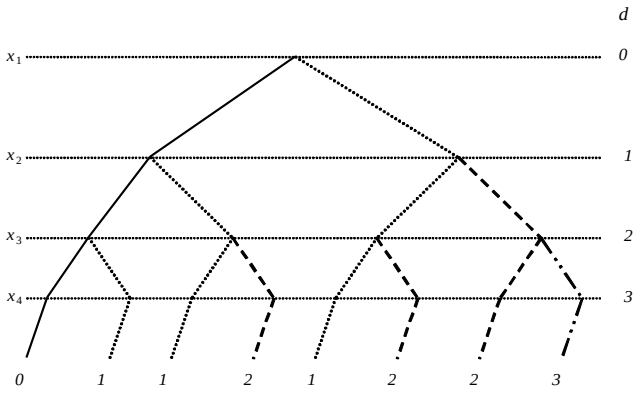
As an example to illustrate the above exploration processes, let us consider a decision problem (in minimization form of an objective function *f()*) consisting of four binary variables x_1, x_2, x_3, x_4. The value ordering heuristic orders nodes left to right and, by convention, we consider that we 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. Initially the

reference solution S_{ref} (associated to criterion value f_{ref}) is the left branch of the tree (thick line in Figure 2).

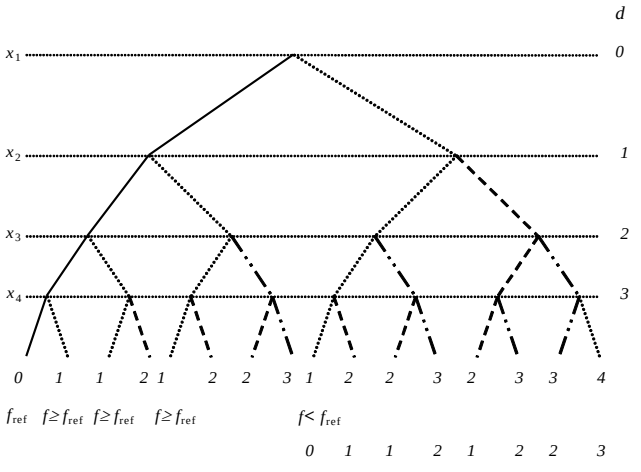
Figure 2 illustrates the search trees obtained using LDS (a), DDS (b), and CDS (c). In this example for DDS, discrepancies are authorized till depth 2. The subscribed number below branches corresponds to the discrepancy number. The types of dotted lines are representative of the successive iterations.



a) LDS



b) DDS ($d=2$)



c) CDS

Figure 2. LDS, DDS, and CDS search principles

3.3. The proposed approach

3.3.1. Problem variables and constraints

To solve the FJSP under study, we have to select an operation, to allocate a resource for the selected operation, to set its start time, and so on until scheduling all operations. To reduce the makespan, we only consider two kinds of variables: operation selection and resource allocation. (The start time of each operation will be set at the earliest possible value.) The values of these two types of variables are ordered following a given instantiation heuristic presented below.

We denote by X the operation selection variables vector and by A the resource allocation variables vector. Thus, X_j corresponds to the j^{th} operation in the sequence and A_j is its assignment value ($\forall j = 1,\ldots, NO$, with NO the total number of operations). The domain of variable X_j is $\{O_{1,1}, O_{1,2},\ldots, O_{1,n_1}, O_{2,1},\ldots, O_{N,1},\ldots, O_{N,n_N}\}$ which corresponds to the choice of the operation to be scheduled. The values taken by the X_j variables have to be all different. The A_j domains are $\{M_1,\ldots, M_m\}$. Moreover, we consider precedence constraints between two consecutive operations of the same job and precedence constraints that may join each operation with other jobs operations. The exploration strategy first considers an operation selection variable to choose an operation, secondly considers a resource allocation variable to assign the selected operation to a resource.

3.3.2. Initial solution

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

- *Type 1: Operation selection.* A sensible choice of the operations order seems to be of main interest for obtaining good quality solutions. It has been experimentally proved that it is a good policy to give the priority to the operation having the Earliest Starting Time (*EST*). The starting time of each operation $O_{i,j}$ is calculated as follows:

$$st_{i,j} = \begin{cases} 0 & \textbf{if } j = 1 \\ & \text{with } i=1,\ldots,N \text{ and } j=2,\ldots,n_i \\ C_{i,j} & \textbf{otherwise} \end{cases}$$

where $C_{i,j-1}$ corresponds to the completion time of operation $O_{i,j-1}$ and is equal to $st_{i,j-1}$ + $p_{i,j-1,k}$ where $p_{i,j-1,k}$ is the processing time of $O_{i,j-1}$ on machine $k \in M_{i,j-1}$.

In case of ties, we suggest selecting the operation having the Earliest Due Date (*EDD*). The due date of each operation $O_{i,j}$ (hereafter denoted by $d_{i,j}$) is calculated as follows:

$$d_{i,j} = \begin{cases} \text{UB} & \textbf{if } j = n_i \\ & \text{with } i=1,\ldots,N \text{ and } j=1,\ldots,n_i-1 \\ d_{i,j+1} - Av(p_{i,j+1}) & \textbf{otherwise} \end{cases}$$

where UB is an approximate value of the makespan and $Av(p_{i,j+1})$ is the average processing time of $O_{i,j+1}$ through all machines of $M_{i,j+1}$.

- *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; hence, following an Earliest Completion Time (*ECT*) rule. This latter rule is dynamic, that is, the machine with the highest priority depends on the machines previously loaded. According to discrepancy principle, a rank of machine assignments is specified for each operation, ordering by the *ECT* on each machine.

After each instantiation of Type 2, 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. Moreover, as said before, Type 2 assignment is interleaved with Type 1 selection, that is, a machine is assigned after each operation selection.

3.3.3. Discrepancy for flexible job shop problems

Because of the existence of two types of variables, we consider here two types of discrepancies: discrepancy on operation selection variables and discrepancy on resource allocation variables. Indeed, our goal is to improve the makespan of our solutions, and since

resources are not identical, discrepancy on allocation variables can improve it. Also, the sequence of jobs to be scheduled has an impact on the total completion time.

Therefore, achieving a discrepancy consists in:

- *Selecting another operation to be scheduled than the operation given uppermost by a value ordering heuristic*. Operation 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. Let consider 3 operations and let consider that the first selected operation is O_1, O_2 the second, and O_3 the third one (this order is given by a value ordering heuristic). Selecting another operation than O_1 in the first position (X_1) consists in making one discrepancy in this level, and so on (see Figure 3).

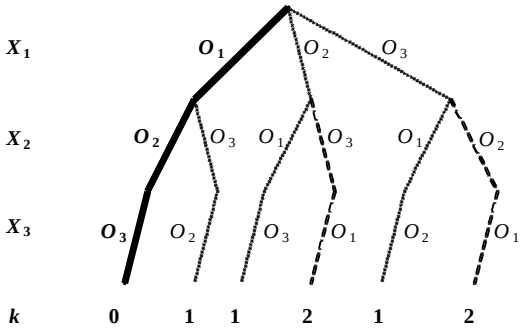


Figure 3. Discrepancies only on the three first operation selection variables

Based on the example above, we note that the search heuristic gives priority to O_1, then to O_2 and finally to O_3 (Sol={1,2,3}), the 1-discrepancy solutions are {2,1,3}, {3,1,2} and {1,3,2}, where the underlined value represents the made discrepancy. A 2-discrepancy solution consists in making an additional discrepancy on the 1-discrepancy ones. Thus, we obtain, {2,3,1} from {2,1,3}, and {3,2,1} from {3,1,2}.

- *Assigning the operation to another resource than the resource suggested by a value ordering heuristic*. 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. In this case, let consider that O_1 can be processed by one machine among the set $\{R_2, R_1, R_3\}$ (this order is given by a value ordering heuristic), O_2 by one machine among $\{R_1, R_4\}$, and O_3 by only R_1. Selecting another machine than R_2 for the first operation O_1 consists in making a discrepancy in this level, and so on (see Figure 4).

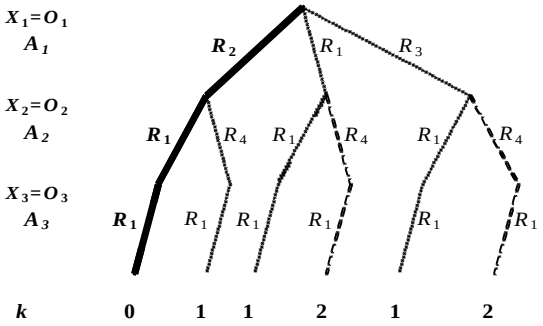


Figure 4. Discrepancies only on the three first resource allocation variables

To obtain solutions of $k + 1$ discrepancies directly from a set of solutions with k discrepancies (without revisiting solutions with $0,\ldots, k-1$ discrepancies), we consider for each solution the latest instantiated variable having the k^{th} discrepancy value and we just have to choose a remaining variable for the $k + 1^{th}$ discrepancy value. This strategy allows us to generate the solutions without redundancy, given that choices already made at the level of a given node will be inhibited during the following choice.

3.3.4. Proposed discrepancy-based method

In our problem, the initial leaf (with 0 discrepancy) is a solution since we do not constrain the makespan value. Nevertheless, we may use the discrepancy principle to expand the search tree for visiting the neighborhood of this initial solution. Two casual ways to stop this exploration are to set a limit for the CPU time or to reach a given lower bound on the makespan. To limit the search tree, one can also use the idea of DDS method that gives priority to variables at the top of the tree (job selection at the initial stages).

To improve the search, we have to consider the CDS method that 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 CDS algorithm to limit the search tree expansion. Therefore, we use *Climbing Depth-bounded Discrepancy Search* (CDDS) [5] which combines both the CDS and the DDS methods. With this method, one can restrict neighborhoods to be visited by only using discrepancies on variables at the top of the tree (see Algorithm 3). Note that DDS algorithm has not been implemented in this paper.

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

Algorithm 3. Climbing Depth-bounded Discrepancy Search

In CDDS algorithm, the function **Compute_Leaves()** generates leaves at discrepancy *k* from the reference solution and at *d*-depth value from the top of the tree, and stops when a better solution than the reference one is reached or when its neighborhood is entirely expanded.

Note that the CDDS method that is discussed in this paper shares the same general framework with the method implemented in [5] for solving a hybrid flow shop problem. However, we have tailored the different search and discrepancy heuristics to capture the specific features of the FJSP.

3.3.4. Proposed neighborhood structures

To diversify solutions, we propose four different types of neighborhood structures that are both based on the block concept [6]. A block (denoted by B) is a succession of at least two critical operations processed on the same resource. A critical operation is an operation belonging to the critical path in the constraint network, which can be defined as the longest schedule. The critical path's operations are connected by a precedence constraint or by a

disjunctive precedence (being able to be realized by the same resource). We cannot obtain an infeasible solution since we respect the precedence and disjunctive constraints.

An interesting property proposed by Jurisch [6] is that if we have two solutions y and y' with $C_{max}(y') < C_{max}(y)$, then we necessarily get at least one of these properties:

- In y', there is at least one operation of a block B of y having been processed on a different resource than in y.
- In y', there is at least one operation of a block B of y, different from the first operation, processed before the first one of B.
- In y', there is at least one operation of a block B of y, different from the last operation, processed after the last one of B.

Based on these all Jurisch's properties, we can obtain two types of neighborhoods $N1$ and $N2$ presented as follows.

- The first neighborhood $N1$ consists in:
    - Assigning each operation O_i of B, on another machine R available at the release date of O_i (denoted by r_i). If R is occupied by O_j at r_i, then we give priority to O_i (*i.e.*, O_i precedes O_i, denoted by $O_i \rightarrow O_j$).
    - Sequencing an operation of B, different from the first (resp. the last) one, before (resp. after) the first (resp. the last) operation of B. Only an adjustment of the starting time of operations will be considered.
- The second neighborhood $N2$ consists in:
    - Assigning each operation O_i of B, on another machine R available at r_i. If R is occupied by O_j at r_i, then we assign O_j on another machine given by ECT.
    - Sequencing an operation of B, different from the first (resp. the last) one, before (resp. after) the first (resp. the last) operation of B. An adjustment of successor's operations assignment and starting time will

be done using ECT.

- Two natural extensions of previous neighborhoods consist in trying *all* possible insertion positions for each considered operation. Hence, extending $N1$ yields a third neighborhood $N3$ that is defined as:

    - Assigning each operation O_i of B, on another machine R available at r_i. If R is occupied by O_j at r_i, then we give priority to O_i ($O_i \rightarrow O_j$).
    - Sequencing an operation of B, different from the first (resp. the last) one, before (resp. after) *each* operation of B. Only an adjustment of the starting time of operations will be considered.

Also, extending $N2$ yields a fourth neighborhood $N4$ that is defined as:

- Assigning each operation O_i of B, on another machine R. available at r_i. If R is occupied by O_j at r_i, then we assign O_j on another machine given by ECT.
- Sequencing an operation of B, different from the first (resp. the last) one, before (resp. after) *each* operation of B. An adjustment of successor's operations assignment and starting time will be done using ECT.

We note that $N3$ dominates $N1$ ($N1 \subseteq N3$) and that $N4$ dominates $N2$ ($N2 \subseteq N4$).

These neighborhood structures imply that discrepancies are not applied to all problem variables. Only some relevant ones, chosen by using the block notion, will be considered. Therefore, using neighborhood structures defined above prunes the process of discrepancies by focusing on promising chosen variables with regard to the considered evaluation criterion. Indeed, it has been experimentally selected that using a complete-swap neighborhood leads to a lesser efficiency both in terms of quality and of CPU time (see Section 4.2). Note that a complete-swap neighborhood structure implies that discrepancies are applied to all problem variables. Hence, two neighborhood structures are developed and defined as follows:

- The first complete-swap neighborhood $CS-N1$ consists in:

- Assigning each considered operation O_i, on another machine R available at r_i. If R is occupied by O_j at r_i, then we give priority to O_i ($O_i \rightarrow O_j$).
- Sequencing a considered operation, different from the first (resp. the last) one, before (resp. after) *each* operation in the sequence respecting the precedence and disjunctive constraints. Only an adjustment of the starting time of operations will be considered.

- The second complete-swap neighborhood $CS-N2$ consists in:
  - Assigning each considered operation O_i, on another machine R available at r_i. If R is occupied by O_j at r_i, then we give priority to O_i ($O_i \rightarrow O_j$).
  - Sequencing a considered operation, different from the first (resp. the last) one, before (resp. after) *each* operation in the sequence respecting the precedence and disjunctive constraints. An adjustment of successor's operations assignment and starting time will be done using ECT.

The next section is devoted to the evaluation of CDDS method via the four developed neighborhoods for the flexible job-shop scheduling problem.

4. Computational experiments

4.1. Test beds

We propose to compare four different strategies ($CDDS-N1$, $CDDS-N2$, $CDDS-N3$, and $CDDS-N4$) in terms of solutions quality and CPU time. Also, we report a comparison between our method and state-of-the-art algorithms. We considered four types of benchmark instances presenting as follows:

- The first data set (*BRdata*) is a set of 10 problems from Brandimarte [8]. The number of jobs ranges between 10 and 20, and the number of machines ranges between 4 and 15. The flexibility per operation ranges between 1.43 and 4.10. Machines are unrelated.

- The second data set (*BCdata*) is a set of 21 problems from [10]. As a job shop can be extended to a flexible job shop by simply adding some machines that already exist in the job shop, the data were constructed from three of the most challenging classical job shop scheduling problems [24, 25] (mt10, la24, la40) by replicating machines. The processing times for operations on replicated machines are assumed to be identical to the original. The number of jobs ranges between 10 and 15, and the number of machines varies between 11 and 18. The flexibility per operation varies between 1.07 and 1.30. Machines are identical.
- The third test sample (*DPdata*) is a set of 18 problems from [11]. The flexibility per operation varies between 1.13 and 5.02. Machines are identical for 50% of instances, and unrelated for the others.
- The fourth test sample (*HUdata*) is a set of 129 problems from [9]. The problems were obtained from three problems by Fisher and Thompson [24] and 40 problems by Lawrence [25] (la01 – la40). Hurink *et al.* [9] generated three sets of test problems: Edata, Rdata and Vdata. The first set contains the problems with the least amount of flexibility (1.15), whereas the average flexibility is equal to 2 in Rdata and to *m*/2 in Vdata (ranges between 2.50 and 7.50). Machines are identical.

The proposed CDDS algorithm was 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 to 15 s for all test instances except for DPdata instances. For such DPdata instances, we set this maximum CPU time to 200 s because of the huge values of operation processing times that imply significant additional CPU time to find solutions. If no optimal solution was found within time limit, then the search is stopped and the best solution is output as the final schedule. The depth of discrepancy in our method is fixed at 7 from the top of the tree; this number has been experimentally chosen.

4.2. Computational results

The results of the computational study on BRdata are summarized in Table 1. Results given by each algorithm are compared with LBs proposed in [2]. The first and second columns symbolize the name and size of the problem, respectively. In the third column, the average number of available machines per operation (*flex.*) is shown for each problem. In the fourth column, (LB,UB) denotes the best lower and upper bounds that have ever been found (a

unique value means that LB=UB; the optimum value is known). The bold values indicate that the corresponding algorithm found the best among the four approaches. The total number of the best solutions given by each algorithm is summarized in the line *#Best*. The values underlined in bold indicate optimal solutions ($C_{\max} = LB$). The makespan associated with an asterisk is the best-known upper bound so far. The Mean Relative Error (MRE) is calculated as follows: $MRE = \frac{(C_{\max} - LB)}{LB} \times 100\ \%$.

As shown in Table 1, $CDDS - N4$ produces the best gap which is equal to 15.03% and generates 90% of best found solutions, followed by $CDDS - N2$, which presents 15.27% as a gap and 90% of best found solutions. In the third place comes $CDDS - N3$ which presents an average gap equal to 15.33% and provides 80% of best found solutions, and finally comes $CDDS - N1$ with 15.76% as a gap from LBs and 70% of best found solutions. We can conclude on the contribution of the use of the dynamic heuristic ECT which allows pushing the search in a more effective way. We also note that $N3$ dominates $N1$ for all instances in 15 seconds, while $N4$ dominates $N2$ for only 9 instances. If we consider all approaches, CDDS presents an average gap of 14.98% from lower bounds. When a complete-swap neighborhood is considered, CDDS presents an average gap of 17.59% from lower bounds. Thus, we can conclude on the contribution of the use of block notion which permits boosting the search in a more effective way.

Table 1. Performance of each approach of CDDS on BRdata

Instance	**N×m**	***flex.***	**(LB,UB)**	***CDDS***			
				$N1$	$N2$	$N3$	$N4$
MK01	10×6	2.09	(36,42)	**40**	**40**	**40**	**40**
MK02	10×6	4.10	(24,32)	**26**	**26**	**26**	**26**
MK03	15×8	3.01	(204,211)	**204**	**204**	**204**	**204**
MK04	15×8	1.91	(48,81)	**60**	**60**	**60**	**60**
MK05	15×4	1.71	(168,186)	175	**173**	**173**	**173**
MK06	10×15	3.27	(33,86)	60	59	59	**58**
MK07	20×5	2.83	(133,157)	**139**	**139**	**139**	**139**
MK08	20×10	1.43	523	**523**	**523**	**523**	**523**
MK09	20×10	2.53	(299,369)	**307**	**307**	**307**	**307**
MK10	20×15	2.98	(165,296)	198	**197**	198	198
MRE				*15.76*	*15.27*	*15.33*	*15.03*
#Best				*7*	*9*	*8*	*9*

A comparison in terms of CPU time between the four strategies is not significant considering this class of problems, since 80% of the problems required the maximum CPU time (15 s), and for the two other instances (MK03 and MK08), we obtain the optimal solutions at the initial solutions (≈ 0 s).

Next, in Table 2 we compare our method CDDS in terms of MRE and time with the genetic algorithm (GA) of Pezzella *et al.* [3], the Tabu Search method (TS) of Mastrolilli and Gambardella [2], and the genetic algorithm (hGA) of Gao *et al.* [4]. Contrarily to CDDS method that is deterministic by nature, TS and hGA algorithms are carried out five runs on the same problem instance in order to obtain meaningful results. CDDS results are obtained through the four above-described different deterministic neighborhood structures. Thus, a comparison of the respective CPU times is irrelevant.

Table 2. Performance comparison between the proposed CDDS and (GA, TS, hGA) on BRdata

Instance	**N×m**	***flex.***	**(LB,UB)**	***CDDS***		***GA***		***TS***		***hGA***	
				C_{max}	***MRE***	C_{max}	***MRE***	C_{max}	***MRE***	C_{max}	***MRE***
MK01	10×6	2.09	(36,42)	**40*** **(40)**	11.11 (11.11)	**40**	11.11	**40** **(40)**	11.11 (11.11)	**40** **(40)**	11.11 (11.11)
MK02	10×6	4.10	(24,32)	**26***	8.33	**26**	8.33	**26**	8.33	**26**	8.33
MK03	15×8	3.01	(204,211)	**204***	0.00	**204**	0.00	**204**	0.00	**204**	0.00
MK04	15×8	1.91	(48,81)	**60***	25.00	**60**	25.00	**60**	25.00	**60**	25.00
MK05	15×4	1.71	(168,186)	173	2.98	173	2.98	173	2.98	**172***	2.38
MK06	10×15	3.27	(33,86)	**58***	75.76	63	90.91	**58**	75.76	**58**	75.76
MK07	20×5	2.83	(133,157)	**139***	4.51	**139**	4.51	144	8.27	**139**	4.51
MK08	20×10	1.43	523	**523***	0.00	**523**	0.00	**523**	0.00	**523**	0.00
MK09	20×10	2.53	(299,369)	**307***	2.68	311	4.01	**307**	2.68	**307**	2.68
MK10	20×15	2.98	(165,296)	**197***	19.39	212	28.48	198	20.00	**197**	19.39
MRE				14.98 (15.34)		17.53		15.41 (15.83)		14.92 (14.92)	
#Best				9 (7)		6		7 (6)		10 (10)	
CI-CPU				385 (96)		unknown		370 (74)		455 (91)	

This comparison is dressed respectively on BRdata. Columns 1 to 4 correspond to the same quantities of Table 1. For each instance, we give the best makespan resulted from the corresponding algorithm (out of four strategies of CDDS algorithm and five runs for GA, TS, and hGA algorithms) and the average makespan under it, besides of the Mean Relative Error (MRE) of the obtained results. Line #Best gives the number of solutions having the best-known upper bound so far. Line CI-CPU gives the sum of the computer-independent CPU time and the average time under it, in seconds, out of four (resp. five) runs for CDDS (resp. GA, TS and hGA) to compute all the solutions of each data set. These values were computed using the normalization coefficients of Dongarra [26]. Thus, to normalize CDDS CPU time values with ones of Gao *et al.* [4], we have multiplied CDDS CPU time by 0.803=(2426/3018). Referred to Dongarra's conversion tables, our machine spends 2426 Mflop/s while the Gao *et al.*'s ones spends 3018 Mflop/s. As mentioned above, a comparison in terms of CPU time is irrelevant because of the deterministic nature of CDDS faced to GA, hGA and TS methods, but we give these values to just have an idea about the speed of each method.

We observe that CDDS outperforms the GA algorithm of Pezzella *et al.* [3] and the TS algorithm of Mastrolilli and Gambardella [2] as well. It also remains comparable with the best-known GA of *Gao et al.* [4]. Indeed, hGA outperforms CDDS on only one instance (MK05). When all instances are considered, the mean relative error of CDDS is 14.98% faced to 14.92% for hGA.

For all next instances, Pezzella *et al.* [3] have reported only the MRE of the best-obtained schedules without the CPU time used and without details of each instance.

Table 3 displays a comparison between the different variants of CDDS and other best-known algorithms on BCdata. Globally, CDDS found 81% of best known-solutions (17 among 21 test instances), while TS found 86% (18 among 21 test instances) and hGA found 62% (13 among 21 test instances). We note that TS algorithm outperforms CDDS on only one instance (mt10xyz). Besides, the average makespan out of four runs of CDDS is little better than all other algorithms. Indeed, average MRE of average makespan of CDDS is equal to 22.60% faced to 22.63% for TS and 22.66% of hGA method.

Table 3. Performance comparison between the proposed CDDS and (TS, hGA) on BCdata

Instance	N×m	*flex.*	(LB,UB)	*CDDS*					*TS*	*hGA*
				N1	*N2*	*N3*	*N4*	C_{max}		
mt10x	10×11	1.10	(655, 929)	**918**	**918**	**918**	**918**	918* (918)	918 (918)	918 (918)
mt10xx	10×12	1.20	(655, 929)	**918**	**918**	**918**	**918**	918* (918)	918 (918)	918 (918)
mt10xxx	10×13	1.30	(655, 936)	**918**	**918**	**918**	**918**	918* (918)	918 (918)	918 (918)
mt10xy	10×12	1.20	(655, 913)	**906**	**906**	**906**	**906**	906 (906)	906 (906)	905* (905)
mt10xyz	10×13	1.20	(655, 849)	851	851	851	**849**	849 (850.5)	847* (850)	849 (849)
mt10c1	10×11	1.10	(655, 927)	930	**928**	**928**	**928**	928 (928.5)	928 (928)	927* (927.2)
mt10cc	10×12	1.20	(655, 914)	912	**910**	911	**910**	910* (910.75)	910 (910)	910 (910)
setb4x	15×11	1.10	(846, 937)	**925**	**925**	**925**	**925**	925* (925)	925 (925)	925 (931)
setb4xx	15×12	1.20	(846, 930)	**925**	**925**	**925**	**925**	925* (925)	925 (926.4)	925 (925)
setb4xxx	15×13	1.30	(846, 925)	**925**	**925**	**925**	**925**	925* (925)	925 (925)	925 (925)
setb4xy	15×12	1.20	(845, 924)	**916**	**916**	**916**	**916**	916* (916)	916 (916)	916 (916)
setb4xyz	15×13	1.30	(838, 914)	908	908	**905**	**905**	905* (906.5)	905 (908.2)	905 (905)
setb4c9	15×11	1.10	(857, 924)	**919**	**919**	**919**	**919**	919 (919)	919 (919.2)	914* (914)
setb4cc	15×12	1.20	(857, 909)	911	911	**909**	911	909* (910.5)	909 (911.6)	914 (914)
seti5x	15×16	1.07	(955, 1218)	**1201**	1203	**1201**	**1201**	1201* (1201.5)	1201 (1203.6)	1204 (1204)
seti5xx	15×17	1.13	(955, 1204)	**1199**	**1199**	**1199**	**1199**	1199* (1199)	1199 (1200.6)	1202 (1203)
seti5xxx	15×18	1.20	(955, 1213)	1198	1198	**1197**	**1197**	1197* (1197.5)	1197 (1198.4)	1204 (1204)
seti5xy	15×17	1.13	(955, 1148)	1140	1137	1139	**1136**	1136* (1138)	1136 (1136.4)	1136 (1136.5)
seti5xyz	15×18	1.20	(955, 1127)	**1125**	1126	**1125**	**1125**	1125* (1125.3)	1125 (1126.6)	1126 (1126)
seti5c12	15×16	1.07	(1027,1185)	**1174**	1175	**1174**	1175	1174* (1174.5)	1174 (1174.2)	1175 (1175)
seti5cc	15×17	1.13	(955, 1136)	1138	1137	**1136**	1137	1136* (1137)	1136 (1136.4)	1138 (1138)
MRE				22.65	22.62	22.58	22.58	22.54 (22.60)	22.53 (22.63)	22.61 (22.66)
#Best				11	9	15	14	17 (8)	18 (7)	13 (10)
CI-CPU				253	253	253	253	1012 (253)	1780 (356)	4105 (821)

Table 4. Performance comparison between the proposed CDDS and (TS, hGA) on DPdata

Instance	N×m	*flex.*	(LB,UB)	*CDDS*					*TS*	*hGA*
				N1	*N2*	*N3*	*N4*	C_{max}		
01a	10×5	1.13	(2505, 2530)	**2518**	2530	2530	2520	2518* (2524.5)	2518 (2528)	2518 (2518)
02a	10×5	1.69	(2228, 2244)	**2231**	2244	2232	**2231**	2231* (2234.5)	2231 (2234)	2231 (2231)
03a	10×5	2.56	(2228, 2235)	**2229**	2235	2230	2233	2229* (2231.8)	2229 (2229.6)	2229 (2229.3)
04a	10×5	1.13	(2503, 2565)	2510	2520	2507	**2503**	**2503*** (2510)	2503 (2516.2)	2515 (2518)
05a	10×5	1.69	(2189, 2229)	2220	2219	**2216**	2217	2216* (2218)	2216 (2220)	2217 (2218)
06a	10×5	2.56	(2162, 2216)	2199	2214	2201	**2196**	2196* (2202.5)	2203 (2206.4)	2196 (2198)
07a	15×8	1.24	(2187, 2408)	2299	**2283**	2293	2307	2283* (2295.5)	2283 (2297.6)	2307 (2309.8)
08a	15×8	2.42	(2061, 2093)	**2069**	**2069**	**2069**	**2069**	2069* (2069)	2069 (2071.4)	2073 (2076)
09a	15×8	4.03	(2061, 2074)	2069	**2066**	**2066**	**2066**	2066* (2066.8)	2066 (2067.4)	2066 (2067)
10a	15×8	1.24	(2178, 2362)	2301	**2291**	2307	2311	2291* (2302.5)	2291 (2305.6)	2315 (2315.2)
11a	15×8	2.42	(2017, 2078)	2078	2069	2078	**2063**	2063* (2072)	2063 (2065.6)	2071 (2072)
12a	15×8	4.03	(1969, 2047)	2034	2031	2040	2031	2031 (2034)	2034 (2038)	2030 (2030.6)
13a	20×10	1.34	(2161, 2302)	**2257**	2265	2260	2259	2257* (2260.3)	2260 (2266.2)	2257 (2260)
14a	20×10	2.99	(2161, 2183)	**2167**	2189	2183	2176	2167* (2178.8)	2167 (2168)	2167 (2167.6)
15a	20×10	5.02	(2161, 2171)	2167	**2165**	2178	2171	2165* (2170.3)	2167 (2167.2)	2165 (2165.4)
16a	20×10	1.34	(2148, 2301)	2259	**2256**	2260	**2256**	2256 (2257.8)	2255 (2258.8)	2256 (2258)
17a	20×10	2.99	(2088, 2169)	2143	**2140**	2156	2143	2140* (2145.5)	2141 (2144)	2140 (2142)
18a	20×10	5.02	(2057, 2139)	2137	**2127**	2131	2131	2127* (2131.5)	2137 (2140.2)	2127 (2130.7)
MRE				2.15	2.20	2.27	2.13	**1.94** (2.18)	2.01 (2.24)	2.12 (2.19)
#Best				6	7	3	6	16 (1)	13 (0)	11 (2)
CI-CPU				2891	2891	2891	2888	11560 (2890)	12335 (2467)	31030 (6206)

Based on the results given in Table 4 on DPdata, we note that CDDS outperforms all other best-known strategies in terms of mean relative error which is equal to 1.94% faced to 7.63% for GA, 2.01% for TS, and 2.12% for hGA. Moreover, it provides 89% of best-known solutions (16 among 18 test instances), while TS provides 72% and only 61% of solutions are given by hGA algorithm.

Table 5 summarizes the Mean Relative Error (MRE) of the results obtained by CDDS (we consider the best makespan resulted from four strategies of CDDS algorithm and the average makespan), and those obtained by other best-known algorithms on HUdata.

Table 5. Performance comparison between the proposed CDDS and (TS, hGA) on HUdata

Instance	**N×m**	**Edata (*flex.* = 1.15)**			**Rdata (*flex.* = 2)**			**Vdata (*flex.* ∈ [2.50,7.50])**		
		CDDS	***TS***	***hGA***	***CDDS***	***TS***	***hGA***	***CDDS***	***TS***	***hGA***
mt06/10/20	6×6 10×10 20×5	0.00 (0.05)	0.00 (0.10)	0.00 (0.10)	0.34 (0.47)	0.34 (0.36)	0.34 (0.34)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
la01 – la05	10×5	0.00 (0.73)	0.00 (0.00)	0.00 (0.00)	0.11 (0.28)	0.11 (0.24)	0.07 (0.07)	0.13 (0.23)	0.00 (0.11)	0.00 (0.00)
la06 – la10	15×5	0.00 (0.19)	0.00 (0.00)	0.00 (0.00)	0.03 (0.19)	0.03 (0.08)	0.00 (0.00)	0.00 (0.00)	0.00 (0.03)	0.00 (0.00)
la11 – la15	20×5	0.29 (1.12)	0.29 (0.29)	0.29 (0.29)	0.02 (0.37)	0.02 (0.02)	0.00 (0.00)	0.00 (0.00)	0.00 (0.01)	0.00 (0.00)
la16 – la20	10×10	0.49 (1.15)	0.00 (0.00)	0.02 (0.02)	1.64 (1.90)	1.73 (1.77)	1.64 (1.64)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
la21 – la25	15×10	5.70 (6.27)	5.62 (5.93)	5.60 (5.66)	3.82 (4.26)	3.82 (4.38)	3.57 (3.69)	0.70 (1.01)	0.70 (0.85)	0.60 (0.68)
la26 – la30	20×10	3.96 (4.84)	3.47 (3.76)	3.28 (3.32)	0.66 (0.98)	0.59 (0.76)	0.64 (0.72)	0.11 (0.19)	0.11 (0.18)	0.11 (0.13)
la31 – la35	30×10	0.42 (0.83)	0.30 (0.32)	0.32 (3.32)	0.22 (0.41)	0.09 (0.14)	0.09 (0.12)	0.02 (0.04)	0.01 (0.03)	0.00 (0.00)
la36 – la40	15×15	9.10 (9.88)	8.99 (9.13)	8.82 (8.95)	4.85 (4.97)	3.97 (4.47)	3.86 (3.92)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
MRE		2.32 (2.78)	2.17 (2.18)	2.13 (2.40)	1.34 (1.54)	1.24 (1.36)	1.19 (1.21)	0.12 (0.16)	0.095 (0.13)	0.082 (0.09)
CI-CPU		***CDDS***	5570 (1393)							
		TS	82840 (16568)							
		hGA	353725 (70745)							

We note that our algorithm is stronger with a higher degree of flexibility. Indeed, on Hurink Vdata, it presents a tiny gap (equal to 0.12%).

If we considered all types of problems (Table 6), we note that $CDDS-N4$ and $CDDS-N2$ outperform other strategies in terms of quality of solutions but they spend a little more time. Indeed, they present a mean relative error of 7.46% within about 780 s. This result is not surprising because of the use of the dynamic heuristic *ECT* which provides an additional time to calculate new starting times and new assignments.

Table 6. Summary results of the different variants of CDDS

Instances	***Nb***	***CDDS-N1***		***CDDS-N2***		***CDDS-N3***		***CDDS-N4***	
		MRE	***CPU***	***MRE***	***CPU***	***MRE***	***CPU***	***MRE***	***CPU***
BRdata	10	15.76	96	15.27	96	15.33	96	**15.03**	96
BCdata	21	22.65	253	22.62	253	**22.58**	253	**22.58**	253
DPdata	18	2.15	2891	2.20	2891	2.27	2891	**2.13**	2884
Hurink Edata	43	2.88	358	**2.85**	471	**2.85**	419	3.14	454
Hurink Rdata	43	**1.53**	428	1.59	481	1.59	472	1.73	483
Hurink Vdata	43	**0.15**	480	0.21	488	0.21	479	0.16	507
Average		**7.52**	751	**7.46**	780	**7.47**	768	**7.46**	780

Table 7 displays computational results of the several groups of problems in terms of MRE. The first column reports the data set, the second column reports the number of instances for each class, and the next four columns report the MRE of the best solution obtained by CDDS among the four strategies, GA, TS, and by hGA, respectively, with respect to the best-known lower bound. The last three columns report the deviation between CDDS, GA, TS, and hGA respectively (denoted by *dev*.).

Table 7. Summary results (MRE) of the different algorithms CDDS and (GA, TS, hGA)

Instances	***Nb***	***CDDS***	***GA***	***TS***	***hGA***	***dev. (CDDS,GA)***	***dev. (CDDS,TS)***	***dev. (CDDS, hGA)***
BRdata	10	14.98	17.53	15.14	14.92	-2.55	-0.16	0.06
BCdata	21	22.54	29.56	22.53	22.61	-7.02	0.01	-0.07
DPdata	18	1.94	7.63	2.01	2.12	-5.69	-0.07	-0.18
Hurink Edata	43	2.32	6.00	2.17	2.13	-3.68	0.15	0.19
Hurink Rdata	43	1.34	4.42	1.24	1.19	-3.08	0.10	0.15
Hurink Vdata	43	0.12	2.04	0.095	0.082	-1.92	0.03	0.04

Results of Table 7 show that CDDS method outperforms GA of Pezzella *et al.* [3] for the optimization of all types, TS of Mastrolilli and Gambardella [2] on BRdata and DPdata, and hGA of Gao *et al*. [4] on BCdata and DPdata. But hGA and TS algorithms both worked better than the proposed CDDS algorithm on the HUdata.

Relating to the results obtained from the computational study, it seems that the proposed algorithm can be an effective approach for the flexible job-shop scheduling problem.

5. Conclusion

In this paper, a Climbing Depth-bounded Discrepancy Search (CDDS) method is presented to solve a Flexible Job Shop Scheduling Problem with the objective of minimizing makespan. Our CDDS approach is based on ordering heuristics, involves two types of discrepancies, operation selection and resource allocation, and uses the block notion to build neighborhood structures that define relevant variables on which discrepancies are applied. The test problems are benchmarks used in the literature. Considering the quality of solutions, our results are better than the genetic algorithm of Pezzella *et al*. [3] on all instances, the tabu search of Mastrolilli and Gambardella [2] on BRdata and DPdata, and the hybrid genetic algorithm of Gao *et al.* [4] on BCdata and DPdata. Relating to the results obtained from the computational study, we conclude that CDDS is an effective approach that is comparable with the state of the art.

References

[1] Garey, M.R., Johnson, D.S., Sethi, R., (1976). The complexity of flow shop and job shop scheduling. *Mathematics of Operations Research*, 1:117-129.

[2] Mastrolilli M., Gambardella L.M., (2000). "Effective neighbourhood functions for the flexible job shop problem". *Journal of Scheduling*, 3:3–20.

[3] Pezzella F., Morganti G., Ciaschetti G., (2008). "A genetic algorithm for the flexible job-shop scheduling problem". *Computers & Operations Research*, 35(10):3202–3212.

[4] Gao J., Sun L., Gen M., (2008). "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems". *Computers & Operations Research*, 35(9):2892–2907.

[5] Ben Hmida A., Huguet M.-J., Lopez P., Haouari M., (2007). "Climbing Discrepancy Search for solving the hybrid Flow shop". *European Journal of Industrial Engineering*, 1(2):223–243.

[6] Jurisch B., (1992). "Scheduling jobs in shops with multi-purpose machines". *PhD dissertation*, Fachbereich Mathematik/Informatik, Universitat Osnabruck.

[7] Brucker P., Schlie R., (1990). "Job shop scheduling with multi-purpose machines". *Computing*, 45:369–375.

[8] Brandimarte P., (1993). "Routing and scheduling in a flexible job shop by tabu search". *Annals of Operations Research*, 41:157–183.

[9] Hurink E., Jurisch B., Thole M., (1994). "Tabu search for the job shop scheduling problem with multi-purpose machines". *Operations Research-Spektrum*, 15:205–215.

[10] Barnes J.W., Chambers J.B., (1996). "Flexible job shop scheduling by tabu search". Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP96-09.

[11] Dauzère-Pérès S., Paulli J., (1997). "An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search". *Annals of Operations Research*, 70:281–306.

[12] Yang J-B, (2001). "GA-based discrete dynamic programming approach for scheduling in FMS environments". *IEEE Transactions on Systems, Man, Cybernetics—Part B*; 31(5):824–835.

[13] Kacem, I., Hammadi, S., Borne, P., 2002. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics—Part C*; 32(1):1–13.

[14] Zhang H-P, Gen M., (2005). “Multistage-based genetic algorithm for flexible job-shop scheduling problem”. *Journal of Complexity International*, 11:223–232.

[15] Xia W, Wu Z., (2005). “An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problem”. *Computers & Industrial Engineering*, 48:409–25.

[16] Fattahi P., Saidi Mehrabad M., Jolai F., (2007). “Mathematical modeling and heuristic approaches to flexible job shop scheduling problems”. *Journal of Intelligent Manufacturing*, 18(3):331–342.

[17] Tay J.C., Ho N.B., (2008). “Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems”. *Computers & Industrial Engineering*, 54(3):453–473.

[18] Harvey W.D., (1995). “Nonsystematic backtracking search”. PhD thesis, CIRL, University of Oregon.

[19] Gacias B., Artigues C., Lopez P., (2009). “Parallel machine scheduling with precedence constraints and setup times”. LAAS Research report 09104.

[20] Korf R.E., (1996). “Improved Limited Discrepancy Search”. *Proceedings AAAI-96*, p.286–291.

[21] Walsh T., (1997). “Depth-bounded Discrepancy Search”. *Proceedings IJCAI-97*, p.1388–1395, Nagoya, Japan.

[22] Milano M., Roli A., (2002). “On the relation between complete and incomplete search: an informal discussion”. *Proceedings CPAIOR’02*, p.237–250, Le Croisic, France.

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

[24] Fisher H., Thompson G.L., (1963). “Probabilistic learning combinations of local job shop scheduling rules”. In *Industrial Scheduling*, J.F. Muth and G.L. Thompson (Eds), Englewood Cliffs, NJ: Prentice-Hall. p.225–251.

[25] Lawrence S., (1984). “Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques”. GSIA, Carnegie Mellon University, Pittsburgh, PA.

[26] Dongarra J., (1998). “Performance of various computers using standard linear equations software”. Computer Science Department, University of Tennessee, Knoxville, Tennessee.