

# Climbing Depth-bounded Discrepancy Search for Solving Flexible Job Shop Scheduling Problems

Abir Ben Hmida

LAAS-CNRS, Université de Toulouse, Toulouse, France, and ROI, Ecole Polytechnique de Tunisie,

La Marsa, Tunisia, abenhmid@laas.fr

Marie-José Huguet, Pierre Lopez

LAAS-CNRS, Université de Toulouse, Toulouse, France, {huguet, lopez}@laas.fr

Mohamed Haouari

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

---

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. Experiments have been performed on well-known benchmarks for flexible job shop scheduling.

*Keywords:* Scheduling, Allocation, Discrepancy Search, Flexible Job Shop.

---

## 1. Introduction

The Flexible Job Shop Problem (FJSP) [1,2] is a generalization of the traditional Job Shop scheduling Problem (JSP), in which it is desired to process a set  $J = \{J_1, \dots, J_n\}$  of  $n$  jobs on a set  $M = \{1, \dots, m\}$  of  $m$  machines in the shortest amount of time. Every job  $J_i$  ( $i=1, \dots, n$ ) consists of  $s_i$  operations  $O_{i1}, O_{i2}, \dots, O_{is_i}$  which must be processed in the given order. Every operation  $O_{is}$  must

be assigned to a unique machine  $R$ , selected among a given subset  $M_{is} \subseteq M$  ( $\forall i, \bigcap_{s=1}^{s_i} M_{is}$  might be

nonempty), which must process the operation without interruption during  $p_i^R$  units, and a machine can process at most one operation at a time. The goal is to choose for each operation a suitable machine and a starting time so that the maximum completion time  $C_{\max}$  (the makespan) is minimized. As a generalization of the job shop problem, the FJSP is more complex because of the additional need to determine the assignment of operations to machines. So, this problem is known to be strongly NP-Hard even if each job has at most three operations and there are two machines [3].

Most of the literature on the shop scheduling problem concentrates on the classical JSP case. Relatively recently, the FJSP has captured the interests of many researchers. The first paper that addresses the FJSP was given by Brucker and Schlie [4], which proposes 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. For solving the general case with more than two jobs, two approaches have been used: hierarchical approach and integrated approach. The hierarchical approach is based on the idea of decomposing the original problem in order to reduce its complexity. Brandimarte [5] was the first author to use this decomposition for the FJSP. He solved the assignment problem using some existing dispatching rules and then focused on the resulting job shop subproblems, which are solved using a tabu search heuristic. Mati et al. [6] proposed a greedy heuristic for simultaneously dealing with the assignment and the sequencing subproblems of the flexible job shop model. The advantage of Mati's heuristic is its ability to take into account the assumption of identical machines. Kacem et al. [7] used a genetic algorithm (GA) to solve the FJSP and they adapted two approaches to solve jointly the assignment and the JSP. The first one is to approach by localization. It makes it possible to solve the problem of resource allocation and build an

ideal assignment mode. The second one is an evolutionary approach controlled by the assignment model and applying GA to solve the FJSP.

In this paper, we propose a new discrepancy-based method called *Climbing Depth-bounded Discrepancy Search* (CDDS) to solve the FJSP. Note that CDDS has been first developed to solve Hybrid Flow Shop problems [8] and has proved its efficiency in this domain.

The remainder of this paper is organized as follows. Section 2 introduces the principles of different discrepancy-based methods, as well as their associated algorithms. Section 3 describes the proposed CDDS method while Section 4 presents its performance on Brandimarte's benchmarks. Finally, Section 5 gives some concluding remarks and directions for future work.

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

### 2.1 Limited Discrepancy Search

The objective of LDS proposed by Harvey in [9] 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, this number of changes corresponds to the number of discrepancies from the initial instantiation. The method stops when a solution is found (if such a solution does exist) or when an inconsistency is detected (the tree is entirely expanded).

The concept of discrepancy was first introduced for binary variables. In this case, exploring the branch corresponding to the best Boolean value (according a value ordering) involves no discrepancy while exploring the remaining branch implies 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) 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. In our work, we chose to adopt the first way for the discrepancy-counting.

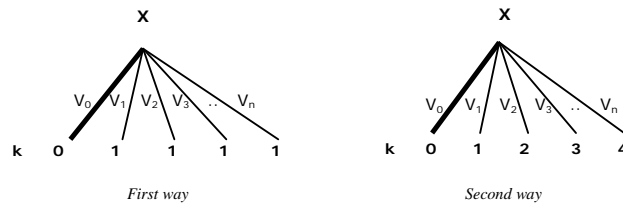


Figure 1. Counting discrepancies

Dealing with a problem defined over  $N$  binary variables, an LDS strategy can be described as shown in Algorithm 1. In such a primal 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 [10]. 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 [11]. 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. These methods can be improved by adding local con-

straint propagation such as Forward Checking [12]. After each instantiation, Forward Checking suppresses inconsistent values in the domain of not yet instantiated variables involved in a constraint with the assigned variable.

---

```

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

```

---

*Algorithm 1. Limited Discrepancy Search*

## 2.2 Climbing Discrepancy Search

CDS is a local search method that adapts the concept of discrepancy to find a good solution for combinatorial optimization problems [13]. It starts from an initial solution suggested by a given heuristic. Hence nodes with discrepancy equal to 1 are first explored then those having a discrepancy equal to 2, and so on. When a leaf with an improved value of the objective function is found, the reference solution is updated, the number of discrepancies is reset to 0, and the process for exploring the neighborhood is again restarted (see Algorithm 2).

---

```

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

```

---

*Algorithm 2. Climbing Discrepancy Search*

The aim of CDS strategy is not only to find a feasible solution, but rather a high-quality solution in terms of criterion value. As mentioned by their authors, the CDS method is close to the Variable Neighborhood Search (VNS) [14]. VNS starts with an initial solution and iteratively explores neighborhoods more and more distant from this solution. The exploration of each neighborhood terminates by returning the best solution it contains. If this solution improves the current one, it becomes the reference solution and the process is restarted. The interest of CDS is that the principle of discrepancy defines neighborhoods as branches in a search tree. Thus, a gradual increase of the allowed discrepancies builds variable-size neighborhoods. The use of a discrepancy-based procedure therefore leads to structure the local search method and then to restrict redundancies.

## 3. The proposed approach

### 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, and to set its start time. 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,  $X_i$  corresponds to the  $i^{\text{th}}$  operation in the sequence and  $A_i$  is its affectation value ( $\forall i=1, \dots, N$ , with  $N$  the number of all operations). The domain of variable  $X_i$  is  $\{O_{11}, O_{12}, \dots, O_{1s_1}, O_{21}, \dots, O_{n1}, \dots, O_{ns_n}\}$  which corresponds to the choice of the operation to be scheduled. The values taken by the  $X_i$  variables have to be all different. The  $A_i$  domains are  $\{1, \dots, m\}$ . Moreover, we consider precedence constraints between two consecutive operations of the same job and precedence constraints that can join each operation with other jobs operations.

### 3.2 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 may have an impact on the total completion time.

Therefore, achieving a discrepancy consists in:

- Selecting another operation to be scheduled than the operation given 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 a discrepancy in this level, and so on (see Figure 2).

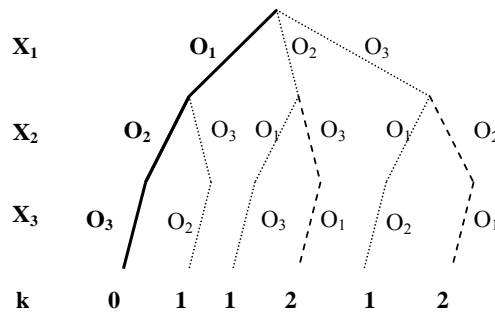


Figure 2. Discrepancies only on the three first operation selection variables.

- 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  $\{R_1, R_4\}$ , 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 3).

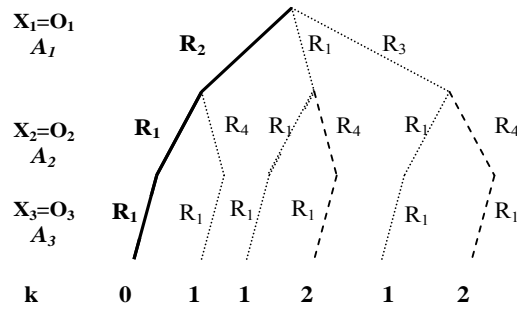


Figure 3. Discrepancies only on the three first resource allocation variables.

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

### 3.3 Instantiation heuristics and propagation

The exploration strategy first consider operation selection variable to choose an operation, secondly consider resource allocation variable to assign the selected operation to a resource. We have two types of value ordering heuristics: the first one ranks operations whilst the second one ranks resources.

*Type 1: Operation selection.* Several priority lists have been used. We first give the priority to the operation with the earliest start time (EST) and, in case of equality, we consider three alternative rules:

- SPT (Smallest Processing Time) rule. It orders the operations in the queue in the ascending order of processing times. When a machine is free, the next operation with the shortest time in the queue will be removed for processing.
- EDD (Earliest Due Date) rule. It gives the priority to the operation which belongs to the job with the earliest due date.
- CJ (Critical Job) rule. It gives the priority to the operation belonging to the job with the longest total duration.

*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.

After each instantiation of Type 2, we use a Forward Checking constraint propagation mechanism 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.

### 3.4 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 tree search for visiting the neighborhood of this initial solution. The only way to stop this exploration is 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 use the DDS method that considers in priority 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 tree search expansion. So, we have developed a new strategy called *Climbing Depth-bounded Discrepancy Search* (CDDS) [8]. With this new method,

one can restrict neighborhoods to be visited by only using discrepancies on variables at the top of the tree (see Algorithm 3).

---

```

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

```

---

*Algorithm 3. Climbing Depth-bounded Discrepancy Search*

The next section is devoted to the evaluation of this algorithm for flexible job shop scheduling.

## 4. Computational experiments

### 4.1 Test beds

We have compared our proposed CDDS method for solving a set of 10 benchmarks instances presenting by Brandimarte in [5]. In [1], all the problems have been solved using a Tabu Search (TS) method.

We propose to solve problems under study by the use of three different strategies for applying discrepancies:

S1- Considering discrepancy only on operation selection variables.

S2- Considering discrepancy only on resource allocation variables.

S3- Join the two kinds of discrepancies. Here we consider the best solution given by S1 as a reference solution to S2.

In our study, we propose to compare these three strategies in terms of solutions quality. Also, we report a comparison between the three heuristics (SPT, CJ and EDD).

Next, we have tested our proposed CDDS method for solving another class of instances presented in [16]. In this class of instances resources are identical. Hence, doing a discrepancy on resource allocation variables does not have any interest. We therefore propose to solve those problems using strategy S1 only. This class of instances contains three sub-classes: the first one, noted Edata, which few number of operations that can be assigned to different machines. The second one, noted Rdata, which most operations can be assigned to a few number of different machines, and the last one, noted Vdata, which each operation can be assigned to many different machines.

We have set a maximum CPU time limit to 1800 s. If no optimal solution was found within 1800 s, then the search is stopped and the best solution of CDDS is output as the final schedule. The depth of discrepancy in our methods varies between 3 and 8 from the top of the tree. We have carried out our tests on a Pentium IV 3.20 GHz with 448 Mo RAM. The CDDS algorithm has been coded using C language and run under Windows XP Professional.

### 4.2. Computational results

In Table 1, for all considered problems, we present the best makespan values obtained by CDDS method among the different value ordering heuristics (SPT, CJ and EDD) and among the different strategies, and the TS algorithm of [1]. We also, present the CPU time of the best obtained makespan values. CDDS returns the best known solutions, denoted by an asterisk (\*), for 60% of the ten instances. If all instances are considered, the average deviation of CDDS algorithm from the best known solutions is 1.4%. Table 1 gives, also, a comparison between the three strategies

(S1, S2, and S3) for discrepancies. We consider the best makespan values obtained by each strategy among the different value ordering heuristics (SPT, CJ, and EDD). The third strategy (S3) always gives better solutions in a fixed running time. This result is not surprising since the latter strategy combines the two types of discrepancies.

<i>problems</i>	<i>Job×Mach</i>	<i>LB</i>	<i>UB</i>	<i>S1</i>	<i>S2</i>	<i>S3</i>	<i>CDDS</i>	<i>Av_CDDS (CPU)</i>	<i>TS</i>	<i>Av_TS (CPU)</i>
<b>mk01</b>	10×6	36	42	42	43	40*	40*	0.5	40	0.01
<b>mk02</b>	10×6	24	32	28	30	26*	26*	12.7	26	0.73
<b>mk03</b>	15×8	204	211	204*	204*	204*	204*	0	204	0.01
<b>mk04</b>	15×8	48	81	69	68	60*	60*	136	60	0.08
<b>mk05</b>	15×4	168	186	181	183	175	175	9.6	173	0.96
<b>mk06</b>	10×15	33	86	67	70	60	60	152	58	3.26
<b>mk07</b>	20×5	133	157	148	160	144*	144*	132	144	8.91
<b>mk08</b>	20×10	523	523	526	531	523*	523*	20	523	0.02
<b>mk09</b>	20×10	299	369	331	344	308	308	89	307	0.15
<b>mk10</b>	20×15	165	296	229	240	216	216	7.03	198	7.69

Table 1. Results on Brandimarte's data.

Table 2 shows computational results over two instance classes. The first column reports the data set, the second column the number of instances for each class, the third column the average number of alternative machines per operation. The next three columns respectively report the deviation percentage of the best solution obtained by TS [1], by CDDS, and by the best known genetic algorithm (GA) solving these problems [19], with respect to the best known lower bound. The table shows that our algorithm is stronger with a higher degree of flexibility (Hurink Vdata). Furthermore, the results show that our algorithm outperforms the best genetic algorithm (still remaining less efficient than TS).

<i>Data set</i>	<i>num</i>	<i>alt</i>	<i>TS (%)</i>	<i>CDDS (%)</i>	<i>GA (%)</i>
<b>Brandimarte</b>	10	2.59	15.4	17.3	17.5
<b>Hurink Edata</b>	43	1.15	2.2	5.3	6.0
<b>Hurink Rdata</b>	43	2	1.2	2.5	4.4
<b>Hurink Vdata</b>	43	4.31	0.1	0.6	2.0

Table 2. Deviation percentage over the best known lower bound

Table 3 presents a comparison between rules (SPT, CJ, and EDD) performances. We consider the best makespan values obtained by each ordering heuristic. The third rule (EDD) always gives better solutions in a fixed running time.

<i>heuristics</i>	<i>%Deviation from LBs</i>		
	<i>SPT</i>	<i>CJ</i>	<i>EDD</i>
<b>Brandimarte</b>	28.2	26.2	17.9
<b>Hurink Edata</b>	16.7	13.9	13.6
<b>Hurink Rdata</b>	13.7	6.9	5.7
<b>Hurink Vdata</b>	3.3	1.2	1.1

Table 3. Performances of search heuristics

## 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 and involves two types of discrepancies: operation selection and resource allocation. The test problems are benchmarks used in the literature. Our results are not better compared with those obtained using a Tabu Search, but in terms of makespan, we

can consider that the CDDS method provides promising results, especially if we consider Hurink's instances. Results show that our algorithm is more efficient with a higher degree of flexibility (Hurink VData). Furthermore, the results show that our algorithm outperforms the best genetic algorithm but it remains less efficient than TS for the two instance classes. The percentage deviations from these latter results are presented.

Developments can still be done to improve the solution's quality of CDDS algorithm. Moreover, other variants of CDDS algorithm may be envisaged for instance by including efficient lower bounds for the FJSP, as well as heuristics for backjumping on promising choice points (see [15]). Other types of problems can be also considered like problems presented in [17,18], which present different properties in terms of total and partial flexibility. Experiments on these latter problems are still in progress.

## References

- [1] M. Mastrolilli, L. M. Gambardella (2000), Effective neighbourhood functions for the flexible job shop problem, *Journal of Scheduling* **3**, 3 – 20.
- [2] R. Vaessens (1995), *Generalized Job Shop Scheduling: Complexity and Local Search*, PhD thesis, Eindhoven University of Technology.
- [3] E. Lawler, J.K. Lenstra, A. Rinnooy Kan, D. Shmoys (1975). Sequencing and scheduling: Algorithms and complexity. *Handbook in Operations Research and Management Sc.* **4**, 115 – 124.
- [4] P. Brucker, R. Schlie (1990), Job shop scheduling with multi-purpose machines, *Computing* **45**, 369 – 375.
- [5] P. Brandimarte (1993), Routing and scheduling in a flexible job shop by tabu search, *AOR* **41**, 157 – 183.
- [6] Y. Mati, N. Rezg, X. Xie (2001), An integrated greedy heuristic for a flexible job shop scheduling problem, *Proceedings IEEE-SMC'01*, 2534 – 2539.
- [7] I. Kacem, S. Hammadi, P. Borne (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), 408 – 419.
- [8] A. Ben Hmida, M.-J. Huguet, P. Lopez, M. Haouari (2006), Adaptation of Discrepancy-based Methods for Solving Hybrid Flow Shop Problems, *Proceedings IEEE-ICSSSM'06*, 1120 – 1125.
- [9] W.D. Harvey (1995), *Nonsystematic backtracking search*, PhD thesis, CIRL, Univ. of Oregon.
- [10] R.E. Korf (1996), Improved Limited Discrepancy Search, *Proceedings AAAI-96*, 286 – 291.
- [11] T. Walsh (1997), Depth-bounded Discrepancy Search, *Proceedings IJCAI-97*, 1388 – 1395.
- [12] R. Haralick, G. Elliot (1980), Increasing tree search efficiency for constraint satisfaction problems, *AI* **14**, 263 – 313.
- [13] M. Milano, A. Roli (2002), On the relation between complete and incomplete search: an informal discussion, *Proceedings CPAIOR'02*, 237 – 250.
- [14] P. Hansen, N. Mladenovic (2001), Variable neighborhood search: Principles and applications, *EJOR* **130**, 449 – 467.
- [15] M.-J. Huguet, P. Lopez, A. Ben Hmida (2004), A limited discrepancy search method for solving disjunctive scheduling problems with resource flexibility, *Proceedings PMS'04*, 299–302.
- [16] E. Hurink, B. Jurisch, M. Thole (1994), Tabu search for the job shop scheduling problem with multi-purpose machines, *Operations Research Spectrum* **15**, 205 – 215.
- [17] S. Dauzère-Pérès, J. Paulli (1997), An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search, *AOR* **70**, 281 – 306.
- [18] J.W. Barnes, J.B. Chambers (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, <http://www.cs.utexas.edu/users/jbc/>.
- [19] F. Pezzella, G. Morganti, G. Ciaschetti (2007), A genetic algorithm for the flexible job-shop scheduling problem, *Computers & Operations Research*, Article in Press, Corrected Proof, <http://dx.doi.org/10.1016/j.cor.2007.02.014>.