Adaptation of Discrepancy-based Methods for Solving Hybrid Flow Shop Problems

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

HAL Id: hal-00137979
https://hal.science/hal-00137979

Submitted on 22 Mar 2007

Adaptation of Discrepancy-based Methods
for Solving Hybrid Flow Shop Problems

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

[1]LAAS-CNRS, Toulouse, France (abenhmid@laas.fr, huguet@laas.fr, lopez@laas.fr)
[2]Unité ROI, Ecole Polytechnique de Tunisie (mohamed.haouari@ept.rnu.tn)
[3]INSA, Toulouse, France

ABSTRACT

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

Keywords: Hybrid Flow Shop, Discrepancy Search Methods, Local Search

1. PROBLEM STATMENT

In this study, we consider the *l*-stage Hybrid Flow Shop (HFS) problem with the objective to minimize the makespan. This problem can be denoted by $Fl(P)\|C_{max}$ [10]. We are especially interested in how adapting discrepancy-based search methods for the HFS problem.

Solving the HFS problem consists in assigning a specific machine to each operation of each job as well as sequencing all operations assigned to each machine. Successive operations of a job have to be processed serially through the *l* stages. Job preemption and job splitting are not allowed. The objective is to find a schedule which minimizes the maximum completion time or makespan defined as the elapsed time from the start of the first operation of the first job at stage 1 to the completion of the last operation of the last job at stage *l*. The HFS problem is NP-Hard as soon as it contains two stages and when there is, at least, more than one machine at a stage [5]. Detailed reviews of the applications and solution procedures of the HFS problems are provided in [6][11][14].

Most of the literature has considered the case of only two stages. In [13] authors presented a case study in a two-stage HFS with sequence-dependent setup time and dedicated machines. For more general cases (more than 2 stages), some authors developed a Branch and Bound (B&B) method for optimizing makespan, which can be used to find optimal solutions of only small-sized problem instances [1]. Later, this procedure has been improved in [17]. In this latter study, several heuristics have been developed to compute an initial

upper bound and a genetic algorithm improves the value of this upper bound during the search. In order to reduce the search tree, new branching rules are proposed in [19]. Global lower bounds are developed in [18] which can be used to measure the quality of heuristic solutions when the optimal solution is unknown. Brah and Loo [2] expanded five better performing standard flow shop heuristics to the HFS case and evaluated them with Santos et al.'s lower bounds. Lower bounds are also defined in [14] based on the single-stage sub-problem relaxation. Another B&B procedure for this problem is proposed by Carlier and Néron in [3]. They proved that their algorithm is more efficient than previous exact solution procedures. Recently, a new heuristic method based on Artificial Immune System (AIS) has been proposed to solve HFS problems [4] and proves its efficiency. Results of AIS algorithm have been compared with Carlier and Néron's lower bounds.

In the next section, we give an overview of discrepancy-based methods. The third section presents how to adapt some of these methods to solve the HFS problem. In Section 4, evaluation of the proposed methods on usual benchmarks are detailed. Finally we report some conclusions and open issues to this work.

2. DISCREPANCY-BASED SEARCH METHODS

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

Limited Discrepancy Search, denoted by LDS, is a branching scheme based on the discrepancy principle. It is instantiated to generate several variants, among them, Depth-bounded Discrepancy Search (DDS) or 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 instantiation heuristics (variable and value ordering). It starts from an initial variable instantiation suggested by a given heuristic and successively explores branches with increasing discrepancies from it, i.e. by changing the instantiation of some variables. This number of changes corresponds to the number of discrepancies from the initial instantiation. The method stops when a solution is found (if it exists) 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 1 discrepancy.

It was then adapted to suit to non-binary variables in two ways. The first one considers that choosing the first ranked value (rank 1) leads to 0 discrepancy while choosing all other ranked values implies 1 discrepancy. In the second way, choosing value with rank r implies $r-1$ discrepancies.

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

```
k ← 0     -- k is the number of discrepancies
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

In such a primal implementation, the main drawback of LDS is to be too redundant: during the search for solutions with k discrepancies solutions with 0 to $k-1$ discrepancies are revisited. To avoid this, Improved LDS method (ILDS) was proposed in [14]. Another improvement of LDS consists in applying discrepancy first at the top of the tree to correct early mistakes in the instantiation heuristic; this is the Depth-bounded Discrepancy Search method (DDS) proposed in [20].

In the DDS algorithm, the generation of leaves with k discrepancies is limited by a given depth.

All these methods (LDS, ILDS, DDS) leads to a feasible solution, if it exists, and are closely connected to an efficient instantiation heuristic. These methods can be improved by adding local constraint propagation such as Forward Checking [8]. After each instantiation, Forward Checking suppresses inconsistent values in the domain of not yet instantiated variables involved in a constraint with the assigned variable.

2.2 Climbing Discrepancy Search

CDS is a local search method which adapts the notion of discrepancy to find a good solution for combinatorial optimization problems [15]. It starts from an initial solution suggested by a given heuristic. Then nodes at discrepancy equal to 1 are explored, then those at discrepancy equal to 2, and so on. When a leaf with an improved value of the objective function is found, the reference solution is updated, the number of discrepancies is reset to 0, and the process for exploring the neighborhood starts again (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 to find only a feasible solution, but 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) [7]. 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. This leads to structure the local search method to restrict redundancies.

3. HOW TO ADAPT DISCREPANCY-BASED METHODS TO SOLVE HYBRID FLOW SHOP

3.1 Problem Variables and Constraints

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

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

3.2 Discrepancy for Hybrid Flow Shop

Despite the fact we have two kinds of variables, we only consider here one kind of discrepancy: *discrepancy on job selection variables*. Indeed, our goal is to improve the makespan of our solutions, and since all resources are identical, discrepancy on allocation variables cannot improve it. Therefore, only the sequence of jobs to be scheduled may have an impact on the total completion time.

Therefore, doing a discrepancy consists in selecting another job to be scheduled than the job given by a value ordering heuristic. Job selection variables are N-ary variables. The number of discrepancy is computed as follows: the first value given by the heuristic corresponds to 0 discrepancy, all the other values correspond to 1 discrepancy (see Figure 1).

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

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

3.3 Instantiation Heuristics and Propagation

Variable ordering follows a *stage-by-stage* policy. The

exploration strategy first consider job selection variable to choose a job, secondly consider resource allocation variable to assign the selected job to a resource.
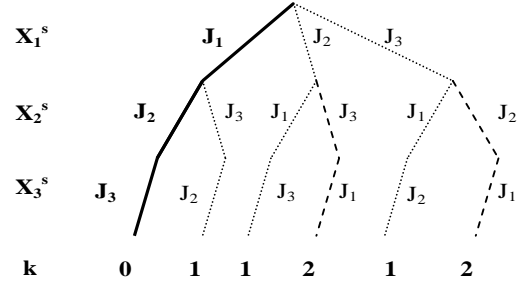

Figure 1. Discrepancies on job selection (stage s)

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

Type 1: job selection. Several heuristics are used. We first give the priority to the job with the earliest start time (*EST*) and in case of equality we consider three kinds of rules: *SPT* (Smallest Processing Time) rule on the first stage, *LPT* (Longest Processing Time) rule on the first stage, and *CJ* (Critical Job) rule. The latter rule gives the priority to the job with the longest 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. This heuristic, called FCT (First Completion Time), is dynamic; the machine with the highest priority depends on the machines previously loaded.

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

3.4 Proposed Discrepancy-based Methods

In our problem, the initial leaf (with 0 discrepancy) is a solution since we do not constrain the makespan value. Nevertheless we may use discrepancy principles to expand the tree search for visiting the neighborhood of this initial solution. The only way to stop this exploration is to fix a limit for the CPU time or to reach a given lower bound on the makespan. To limit the search tree, one can use the *DDS method* which considers in priority variables at the top of the tree (job selection at the first stage). However this kind of method has no guide for searching in a promising space search.

To improve the search, we have to consider the CDS method which goes from an initial solution to a better one and so on. The idea of applying discrepancies only

at the top of the search tree can be also joined with CDS algorithm to limit the tree search expansion. So, we create a new strategy called CDDS method (Climbing Depth-bounded Discrepancy Search). With this new method, one can restrict neighborhoods to be visited by only using discrepancies on variables at the top of the tree (see Algorithm 3).

```
k ←0      -- k is the number of discrepancy
k_max ← N -- N is the number of variables

Sol ← Initial_Solution() -- Sol is the reference
                         -- solution
while (k ≤ k_max) do
    k ← k +1
    -- Generate leaves at discrepancy k from Sol
    -- and at p-depth value from the top of the tree
    -- with 1 < p < N
    Sol'← Compute_Leaves(Sol, p, 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

4. EXPERIMENTS

4.1 Test beds

We compare our adaptation of the DDS method and our proposed CDDS method for solving benchmarks problems which are presented in [3]. In [3], all the problems have been solved using a Branch & Bound (B&B) method operating with use of satisfiability tests and time-bound adjustments. They calculated lower bounds (LBs) of the problems and they limited their search within 1600 s.

In our study, we propose to compare our solutions with these LBs. We also run our algorithm within 1600 s. If optimal solution was not found within 1600 s, the search is stopped and the best solution is accepted 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 192 Mo RAM. DDS and CDDS algorithms have been programmed using C language and run under Windows XP Professional.

4.2. Results

In Table 1, for all considered problems, we present the best makespan values ($C_{\max}$) obtained by our DDS and CDDS methods among the three value ordering heuristics (SPT, LPT, and CJ), and the B&B algorithm of [3] within 1600 s. Deviation from LBs is calculated as follows:

$$\% \text{deviation} = \frac{C_{\max}\_best - LowerBound}{LowerBound}$$

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

In [3] some of the problems are grouped as hard problems. Hard problems consist of the c and d types of 10×5 and 15×5 problems. The rest of the problems (all a, b types and 10×10 c type problems) are identified as easy problems. As shown in Table 1, for a and b type problems better results have been found than for c and d type problems. Indeed, the machine configurations have an important impact on problems complexity that affects solution quality [4].

In Table 2, we compare the efficiency of the three methods for easy and hard problems. As it will be noticed from the table, for easy problems, DDS and CDDS algorithms provide better results than B&B, but for hard problems B&B algorithm is better than DDS algorithm. On the other hand, for hard problems, CDDS method obtains better solutions compared to B&B algorithm in terms of deviation value from LBs.

Table 2. Relative efficiency of the three methods

Method	Easy problems	Hard problems
	%deviation	%deviation
B&B	2.2	6.9
DDS	1.4	8.0
CDDS	1.1	5.0

If all problems are considered, the average deviation from LBs for DDS algorithm is 3.58%, while the average deviation of B&B is 3.68%. For CDDS the average is only of 2.32%. On these benchmarks, our CDDS algorithm provides in average the best solutions.

Table 3 presents a comparison between the value ordering heuristics (SPT, LPT, and CJ) efficiency. For both CDS and CDDS methods, the third rule (CJ) always gives better solutions in a fixed running time.

Table 3. Efficiency of value ordering heuristics

heuristics	SPT	LPT	CJ
%deviation	6.3	5.2	2.5

Our discrepancy-based methods (DDS and CDDS) prove their contributions in terms of improvement of the initial makespan. Within 1600 seconds of CPU time the deviation of the initial makespan has been reduced with DDS algorithm by nearly 14.7% for hard problems and 9.7% for easy ones. If we consider all problems, the initial makespan has been reduced with DDS algorithm by nearly 10.4%. For CDDS, the initial makespan reduction is about 14%. This percentage is distributed as 21% for hard problems and 10.4% for easy ones.

Easy problems instances rapidly converge compared with hard ones. They take, for both DDS and CDDS methods, 13 mn in average to obtain all the solutions

for easy problems, while hard problems take 25 mn in average. Since we have not the B&B code, we can not dress a comparison with it in terms of CPU time.

Table 1. Solutions of test problems (bold problems have been identified as hard problems)

Problem	Cmax			LB of Cmax	%deviation		
	DDS	CDDS	B&B		DDS	CDDS	B&B
J10c5a2	88	88	88	88	0.0	0.0	0.0
J10c5a3	117	117	117	117	0.0	0.0	0.0
J10c5a4	121	121	121	121	0.0	0.0	0.0
J10c5a5	122	122	122	122	0.0	0.0	0.0
J10c5a6	110	110	110	110	0.0	0.0	0.0
J10c5b1	130	130	130	130	0.0	0.0	0.0
J10c5b2	107	107	107	107	0.0	0.0	0.0
J10c5b3	109	109	109	109	0.0	0.0	0.0
J10c5b4	122	122	122	122	0.0	0.0	0.0
J10c5b5	153	153	153	153	0.0	0.0	0.0
J10c5b6	115	115	115	115	0.0	0.0	0.0
J10c5c1	**71**	**69**	**68**	**68**	**4.4**	**1.5**	**0.0**
J10c5c2	**76**	**75**	**74**	**74**	**2.7**	**1.4**	**0.0**
J10c5c3	**73**	**72**	**71**	**71**	**2.8**	**1.4**	**0.0**
J10c5c4	**68**	**66**	**66**	**66**	**3.0**	**0.0**	**0.0**
J10c5c5	**79**	**78**	**78**	**78**	**1.3**	**0.0**	**0.0**
J10c5c6	**70**	**70**	**69**	**69**	**1.4**	**1.4**	**0.0**
J10c5d1	**67**	**66**	**66**	**66**	**1.5**	**0.0**	**0.0**
J10c5d2	**75**	**74**	**73**	**73**	**2.7**	**1.4**	**0.0**
J10c5d3	**65**	**64**	**64**	**64**	**1.6**	**0.0**	**0.0**
J10c5d4	**72**	**70**	**70**	**70**	**2.9**	**0.0**	**0.0**
J10c5d5	**68**	**68**	**66**	**66**	**3.0**	**3.0**	**0.0**
J10c5d6	**64**	**63**	**62**	**62**	**3.2**	**1.6**	**0.0**
J10c10a2	158	158	158	158	0.0	0.0	0.0
J10c10a3	151	148	148	148	2.0	0.0	0.0
J10c10a4	150	149	149	149	0.7	0.0	0.0
J10c10a5	148	148	148	148	0.0	0.0	0.0
J10c10a6	147	146	146	146	0.7	0.0	0.0
J10c10b1	163	163	163	163	0.0	0.0	0.0
J10c10b2	158	157	157	157	0.6	0.0	0.0
J10c10b3	169	169	169	169	0.0	0.0	0.0
J10c10b4	159	159	159	159	0.0	0.0	0.0
J10c10b5	165	165	165	165	0.0	0.0	0.0
J10c10b6	165	165	165	165	0.0	0.0	0.0
J10c10c1	117	118	127	113	3.5	4.4	12.4
J10c10c2	117	117	116	116	0.9	0.9	0.0
J10c10c3	118	117	133	98	20.4	19.4	35.7
J10c10c4	122	121	135	103	18.4	17.5	31.1
J10c10c5	131	128	145	121	8.3	5.8	19.8
J10c10c6	108	106	112	97	11.3	9.3	15.5
J15c5a1	178	178	178	178	0.0	0.0	0.0
J15c5a2	165	165	165	165	0.0	0.0	0.0
J15c5a3	132	130	130	130	1.5	0.0	0.0
J15c5a4	156	156	156	156	0.0	0.0	0.0
J15c5a5	164	164	164	164	0.0	0.0	0.0
J15c5a6	179	178	178	178	0.6	0.0	0.0
J15c5b1	170	170	170	170	0.0	0.0	0.0
J15c5b2	152	152	152	152	0.0	0.0	0.0
J15c5b3	157	157	157	157	0.0	0.0	0.0
J15c5b4	149	147	147	147	1.4	0.0	0.0
J15c5b5	166	166	166	166	0.0	0.0	0.0
J15c5b6	175	175	175	175	0.0	0.0	0.0
J15c5c1	**91**	**90**	**85**	**85**	**7.1**	**5.9**	**0.0**
J15c5c2	**98**	**92**	**90**	**90**	**8.9**	**2.2**	**0.0**
J15c5c3	**93**	**93**	**87**	**87**	**6.9**	**6.9**	**0.0**
J15c5c4	**92**	**90**	**90**	**89**	**3.4**	**1.1**	**1.1**
J15c5c5	**84**	**77**	**84**	**73**	**15.1**	**5.5**	**15.1**
J15c5c6	**94**	**93**	**91**	**91**	**3.3**	**2.2**	**0.0**
J15c5d1	**167**	**167**	**167**	**167**	**0.0**	**0.0**	**0.0**
J15c5d2	**92**	**87**	**85**	**82**	**12.2**	**6.1**	**3.7**
J15c5d3	**89**	**83**	**96**	**77**	**15.6**	**7.8**	**24.7**
J15c5d4	**92**	**86**	**101**	**61**	**50.8**	**41.0**	**65.6**
J15c5d5	**87**	**82**	**97**	**67**	**29.9**	**22.4**	**44.8**
J15c5d6	**88**	**84**	**87**	**79**	**11.4**	**6.3**	**10.1**
J15c10a1	236	236	236	236	0.0	0.0	0.0
J15c10a2	203	200	200	200	1.5	0.0	0.0
J15c10a3	198	198	198	198	0.0	0.0	0.0
J15c10a4	225	225	225	225	0.0	0.0	0.0
J15c10a5	182	182	183	182	0.0	0.0	0.5
J15c10a6	201	200	200	200	0.5	0.0	0.0
J15c10b1	222	222	222	222	0.0	0.0	0.0

J15c10b2	187	187	187	187	0.0	0.0	0.0
J15c10b3	222	222	222	222	0.0	0.0	0.0
J15c10b4	221	221	221	221	0.0	0.0	0.0
J15c10b5	200	200	200	200	0.0	0.0	0.0
J15c10b6	219	219	219	219	0.0	0.0	0.0
Average					3.58	*2.32*	*3.68*

5. CONCLUSIONS AND FURTHER WORKS

In this paper two discrepancy-based methods are presented to solve Hybrid Flow Shop problems with minimization of makespan. The first one is an adaptation of Depth-bounded Discrepancy Search (DDS) to suit to the problem under study. The second one, Climbing Depth-bounded Discrepancy Search (CDDS), combines both CDS and DDS. The two methods are based on instantiation heuristics which guide the exploration process towards some relevant decision points able to reduce the makespan. These methods use usual constraint propagation to prune the search tree.

The test problems are benchmarks used in the literature. The percentage deviations from lower bounds are presented. Our results are compared with B&B results. In terms of makespan, we obtain in average better solutions with the proposed CDDS approach.

Another method, a metaheuristic based on Artificial Immune System (AIS), has proved its efficiency for solving Hybrid Flow Shop problems [4]. Experimental results with the same benchmarking problems show that the average deviation of AIS algorithm from LBs is 1.66% while the CDDS deviation is 2.32%. In contrast, CDDS is a simple local search method based on generic tree search principles. Moreover, a solution can be obtained at any time. The notion of discrepancy permits the neighborhood structuring and avoids the redundancies. Since the current implementation is identical to a simple descent method, a further work will consist to design a diversification mechanism.

REFERENCES

[1] Brah S.A., Hunsucker J.L., "Branch and Bound algorithm for the flow shop with multiprocessors", *EJOR* (51) 88-89, 1991

[2] Brah S.A., Loo L.L., "Heuristics for scheduling in a flow shop with multiple processors", *EJOR* (113) 113-122, 1999

[3] Carlier J., Néron E., "An exact method for solving the multiprocessor flowshop", *RAIRO-OR* (34) 1-25, 2000

[4] Engin O., Döyen A., "A new approach to solve hybrid flow shop scheduling problems by artificial immune system", *FGCS* (20) 1083-1095, 2004

[5] Gupta J.N.D., "Two-stage hybrid flowshop scheduling problem", *JORS* (39) 359-364, 1988

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

[7] Hansen P., Mladenovic N., "Variable neighborhood search: Principles and applications", *EJOR* (130) 449-467, 2001

[8] Haralick R., Elliot G., "Increasing tree search efficiency for constraint satisfaction problems", *AI* (14) 263-313, 1980

[9] Harvey W.D., "Nonsystematic backtracking search", *PhD thesis*, CIRL, University of Oregon, 1995

[10] Hoogeveen J.A., Lenstra J.K., Veltman B., "Premptive scheduling in a two-stage multiprocessor flowshop is NP-Hard", *EJOR* (89) 172-175, 1996

[11] Kis T., Pesch E., "A review of exact solution methods for the non-preemptive multiprocessor flowshop problem", *EJOR* (164) 592-608, 2005

[12] Korf R.E., "Improved limited discrepancy search", *Proceedings AAAI-96*, 286-291, 1996

[13] Lin H.T., Liao, C.J., 2003, "A Case Study in a Two-Stage Hybrid Flow Shop with Setup Time and Dedicated Machines", *IJPE* (86) 133-143, 2003

[14] Linn R., Zhang W., "Hybrid flow shop scheduling: a survey", *Comput. Ind. Eng.* (37) 57-61, 1999

[15] Milano M., Roli A., "On the relation between complete and incomplete search: an informal discussion", *Proceedings CPAIOR'02*, 2002

[16] Moursli O., Pochet Y., "A branch and bound algorithm for the hybrid flow shop", *IJPE* (64) 113-125, 2000

[17] Portmann M.C., Vignier A, Dardihac D., Dezalay D., "Branch and Bound crossed with G.A. to solve hybrid flow shops", *IJPE* (43) 27-137, 1992

[18] Santos D.L., Hunsucker J.L., Deal D.E., "Global lower bounds for flow shops with multiple processors", *EJOR* (80) 112-120, 1995

[19] Vignier A., "Contribution à la résolution des problèmes d'ordonnancement de type monogamme, mutimachines (flow shop hybride)", *PhD Thesis*, University of Tours, France, 1997

[20] Walsh T., "Depth-bounded Discrepancy Search", *Proceedings IJCAI-97*, 1388-1395, 1997