Climbing discrepancy search for flowshop and jobshop scheduling with time-lags

Wafa Karoui, Marie-José Huguet, Pierre Lopez, Mohamed Haouari

Climbing discrepancy search for flowshop and jobshop scheduling with time-lags

Wafa Karoui[1,2]

Marie-José Huguet[1]

Pierre Lopez[1]

Mohamed Haouari[2]

[1] *CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France and Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France*

and

[2] *Unité de recherche ROI ; Ecole Polytechnique de Tunisie, 2078 La Marsa, Tunisia*

Abstract

This paper addresses the jobshop and the flowshop scheduling problems with minimum and maximum time lags. To solve this kind of problems, we propose adaptations of Climbing Discrepancy Search (CDS). We study various parameter settings. Computational experiments are provided to evaluate the propositions.

Keywords: Scheduling, jobshop, flowshop, time lags, discrepancy.

---

[1] Email:`wakaroui,huguet,lopez@laas.fr`

[2] Email:`mohamed.haouari@ept.rnu.tn`

1 Introduction

This paper addresses the jobshop and the flowshop scheduling problems with minimum and maximum time lags. The objective is to find a schedule that minimizes the makespan. Different definitions can be associated to time lags constraints. Initially, Mitten [5] proposes this concept. [2] defines it as time between the end of one operation and the start of another. In our case, we speak about two extra constraints added to the jobshop and flowshop problems, linking successive operations of a same job. Time between these operations is bounded by minimum and maximum time lags. These problems can be considered as a generalization of basic problems without time lags (NP-hard in the strong sense). With time lags, problems become at least as difficult as the basic ones. Few methods have been used to solve this type of problems. [1] proposed a memetic algorithm which obtained good results on jobshop instances with null minimum and maximum time lags (no-wait problems). [3] also studied this problem including generalized resource constraint propagation rules and branch-and-bound.

In this paper, we propose adaptations of Climbing Discrepancy Search (CDS) [6], to solve scheduling problems with time lags. The remainder of the paper is organized as follows. Section 2 introduces the principle of CDS and proposed adaptations for the studied problems. Section 3 synthesizes experiments carried out to evaluate the performance of CDS. Finally, Section 4 highlights the conclusions and some further works.

2 Discrepancy and learning for problems with time lags

To solve problems with time lags, we propose a variant of Climbing Discrepancy Search method, a tree search principle for optimization based on discrepancy. This method starts from an initial solution proposed by a given heuristic and tries to improve it by increasing step by step the number of times we do not follow this solution (discrepancy). It then builds a neighborhood around this initial solution. Nodes with a number of discrepancies equal to 1 are first explored, then those having a number of discrepancies equal to 2, and so on. When a leaf with improved value of the objective function is found, the reference solution is updated, the number of discrepancy is reset to 0, and the process for exploring the neighborhood is restarted. To limit the tree search expansion, we put a stop condition as a timeout on the CPU time. To adjust our method to problems under study, we propose various parameter settings: discrepancy position in the search tree, heuristics to generate the initial solution, learning mechanisms based on weights associated to jobs. The discrepancies counting is binary: the heuristic choice corresponds to zero discrepancies, all the other choices correspond to one discrepancy.

2.1 Discrepancy position

We experiment to diverge alternatively, first at the top of the search tree, or first at its bottom. We try also to diverge only in a part of the tree, for example at the top, and to visit the other part without discrepancy at all.

2.2 Heuristic to generate the initial solution

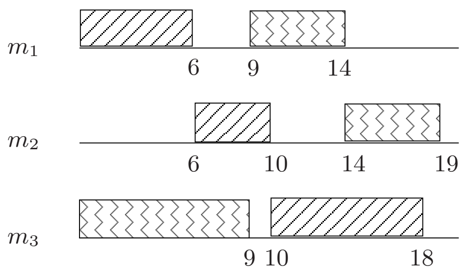
The heuristic selects a job and places all its operations in the order of their definition (routing). In the sequel, we call D the duration of a job equal to the sum of all its operation durations and DTL the sum of all its time lags durations. To obtain an initial solution, we can consider heuristics which sort jobs in the lexicographical order, as in [3], or in the ascending or descending order of D, DTL, D+DTL, and D/DTL. For a given problem, it is obvious that if we start the search from a good solution, we have more chance to get more improvements.

2.3 Learning based on weights on jobs

Learning can guide the method and improve it. To adjust the proposed method to problems under study, we associate a weight to each job. At the beginning, all weights are identical. We can increase the weight $W(J_i)$ associated to job J_i in different ways. We studied three cases as shown in Figure 1 where operations of job J_3 cannot be placed in their first slack period on the associated machine. In Figure 1, (O_{ij}, m_k, d_{ij}) denotes the j^{th} operation of job J_i to execute on machine m_k with duration d_{ij}. $TLmin$ and $TLmax$ denote the minimum time lag and the maximum time lag, respectively.

- Case A) The job weight is increased every time one of its operations is not inserted in some slack period. In the example, the weight of J_3 is then 5.
- Case B) The job weight is increased every time one of its operations is not inserted in the first slack period on one of its associated machine. As we can see in the example, we increment the weight at most one time per machine (or operation). The maximal factor to get on a considered operation is equal to the number of machines (or operations). In the example, the weight of J_3 is then 3.

J_1 and J_2 already scheduled:



$J_1 = \{(O_{11}, m_1, 6), (O_{12}, m_2, 4), (O_{13}, m_3, 8)\}$

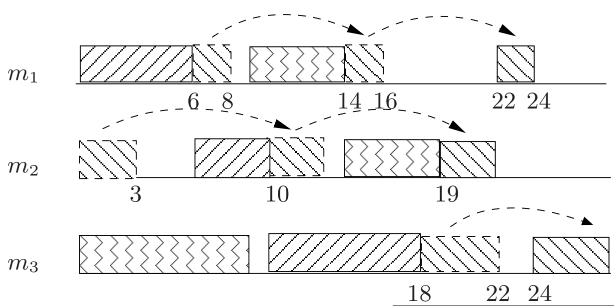
$J_2 = \{(O_{21}, m_3, 9), (O_{22}, m_1, 5), (O_{23}, m_2, 5)\}$

$J_3 = \{(O_{31}, m_2, 3), (O_{32}, m_1, 2), (O_{33}, m_3, 4)\}$

O_{31}-O_{32} : $TLmin$=0 and $TLmax = 1$

O_{32}-O_{33} : $TLmin$=0 and $TLmax = 0$

J_3 insertion:

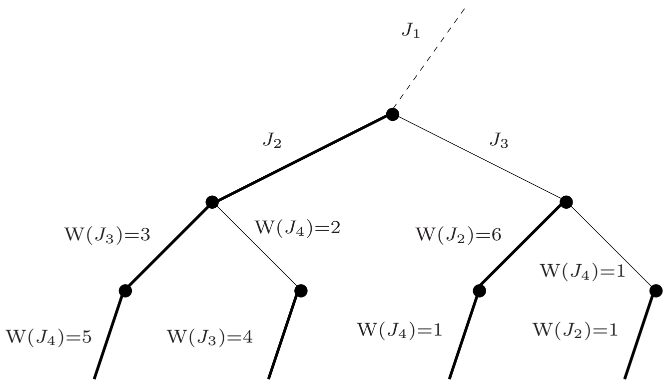


		Case		
		A	B	C
# of incrementations associated to the operation	O_{31}	+2	+1	true
	O_{32}	+2	+1	true
	O_{33}	+1	+1	true
Weights of the job	J_3	+5	+3	+1

Fig. 1. Different ways to increment job weights

- Case C) The job weight is increased every time one or more operations of this job are not placed in their first slack periods on the associated machine. In the example, the weight of J_3 is then 1. As we can see in the example, the weight is increased at most one time for the same job.

Anyway, in addition of the manner to increase weights, we have to choose a way to count them. For example, we can consider the sum of all weights obtained by the job during the iteration (denoted in the following by *Sum*), or the maximum of all its weights (denoted by *Max*) as shown in Figure 2. In the next iterations, obtained weights are integrated in the basic heuristic to choose the job to schedule first. For instance, if the heuristic is based on the duration D of jobs, the heuristic using weights can be based on *D/weights*.



	J_2	J_3	J_4
Max	6	4	5
Sum	7	7	9

Fig. 2. Different ways to count job weights

2.4 Algorithm

Algorithm 1 summarizes the principle of CDS method. Various parameter settings (in bold) offer many choices. The heuristic (line 3) can be adapted, as well as the parameters of CDS iteration (line 6). *Div_pos* increment denotes the discrepancies position chosen to diverge first: either first at the top or first at the bottom. *Case* denotes the case A, B or C chosen to increment the discrepancies. Its value is equal to zero if we do not associate weights to jobs. *Counting* denotes the weight counting way, *Max* or *Sum*.

Algorithm 1 *Climbing Discrepancy Search iteration*

$k \leftarrow 0$ % k is the discrepancy number
$k_{max} \leftarrow n$ % n is the variable number
$S_{init} \leftarrow initial\_solution(\textbf{heuristic})$ % S_{init} is the initial solution
while $k \leq k_{max}$ **do**
    $k \leftarrow k{+}1$ % Generate k-discrepancies branches from S_{init}
    $S_{init}' \leftarrow Generate(S_{init}, k, \textbf{Div\_pos}, \textbf{Case}, \textbf{Counting})$
    if $Best(S'_{init}, S_{init})$ **then**
        $S_{init} \leftarrow S'_{init}$ % Update the initial solution
        $k \leftarrow 0$
    end if
end while

3 Computational results

We experiment our propositions on the data set of classical instances of scheduling problems proposed in [4]. For jobshops, we consider the Lawrence's instances $\{laX\}_{X=1..20}$ in addition to Fisher and Thompson's instances, $ft06$ and $ft10$. For flowshops, we consider Carlier's instances $\{carX\}_{X=5..8}$. The data set contains instances created from classical instances with minimum and maximum time lags generated with a minimum time lag ($TLmin$) equal to 0 and a maximum time lag ($TLmax$) equal to 0, 0.25, 0.5, 1, 2, 3, 5, and 10. Comparisons are done *vs.* results obtained by [1] which consider only the $ft06$ with $TLmax$ of 0, 0.5, 1 and 2, $\{laX\}_{X=1..20}$ with $TLmax$ equal to 0, $\{laX\}_{X=1..5}$ with $TLmax$ equal to 0.5, 1 and 2, and $\{laX\}_{X=6..8}$ with $TLmax$ equal to 0.5, 1, 2 and 10, in addition to $\{carX\}_{X=5..8}$ with $TLmax$ equal to 0, 0.5, 1 and 2. Comparisons are also done *vs.* results obtained with ILOG-Scheduler for all considered instances.

The tests about the heuristic to generate the initial solution show that *descending D* gives the best results. In Table 1, we can observe the number of instances on which every heuristic is the best.

Table 1
Comparison of heuristics to generate an initial solution

Order	D	DTL	D+DTL	D/DTL	Lexicographical
Descending	**55**	51	**55**	29	20
Ascending	5	12	5	22	

For other tests, timeout is of 200 seconds. In general, the best known solutions (*BKSs*) are divided between [1], ILOG-Scheduler, and our propositions, without any regularity. Nevertheless, we claim the following: For the no-wait problems, [1] obtain the best results. For other instances, ILOG-Scheduler provides the best results except for cases referred in Table 2 where our propositions have the best results. In Table 2, *WF* refers to the version of CDS without weights which diverges at the top first. *B-Sum*, respectively *B-Max*, refers to *case B* for weighting jobs, as presented below, associated to the counting way *Sum*, respectively *Max*. Case B seems to be better than other cases of weighting jobs on considered instances.

Table 2
Obtained results on some instances

Instance	*TLmax*	*ILOG-Scheduler*	*WF*	*B-Sum*	*B-Max*
la11	0.25	2058	**1861**	1965	1965
	0.5	1945	**1874**	**1874**	**1874**
la12	0.25	1710	1682	1671	**1656**
la13	0.25	1906	**1897**	1892	1892
	0.5	1804	**1787**	1808	1808
la14	0.25	2143	**1823**	2042	2042
	0.5	2067	1964	**1953**	**1953**
	1	1976	1772	**1762**	**1762**
	2	1976	**1612**	1660	1660
	3	1695	1567	**1542**	**1542**
	5	1695	**1452**	1477	1477
la15	0.25	2371	2084	**2043**	**2043**
	0.5	2217	2118	**1910**	**1910**
la17	0.25	1455	**1410**	1427	1460

4 Conclusions and further works

In this paper, a Climbing Discrepancy Search (CDS) method is proposed to solve jobshop and flowshop scheduling problems with time lags. We studied various parameter settings for the proposed method, such as discrepancy positions, heuristic to generate the initial solution, and learning mechanisms based on weights associated to jobs. Proposed variants were tested on known benchmarks in the literature. The obtained results show that we have to study variants of CDS associated to classical scheduling techniques as heuristic insertion to determine upper bounds, and resource constraint propagation rules adaptation for lower bounds.

References

[1] Caumond, A., Lacomme, P., and Tchernev, N., "A memetic algorithm for the jobshop with timelags," *Computers and Operations Research* **35** (2008), 2331–2356.

[2] Dell'amico, M., "Shop problems with two machines and time lags," *Operations Research* **44(5)** (1996), 777–787.

[3] Huguet, M.-J., Artigues, C., Dugas, M., and Lopez P., "Generalized constraint propagation for solving job shop problems with time lags," *PMS'10*. Submitted.

[4] Lacomme, P., URL: http://www.isima.fr/~ lacomme/Job_Shop_TL.html.

[5] Mitten, L.G., "Sequencing n jobs on two machines with arbitrary time lags," *Management Science* **9** (1958), 293–298.

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