



HAL
open science

Parallel machine scheduling with precedence constraints and setup times

Bernat Gacias, Christian Artigues, Pierre Lopez

► **To cite this version:**

Bernat Gacias, Christian Artigues, Pierre Lopez. Parallel machine scheduling with precedence constraints and setup times. *Computers and Operations Research*, 2010, 37 (12), p. 2141-2151. hal-00362159

HAL Id: hal-00362159

<https://hal.science/hal-00362159>

Submitted on 18 Feb 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Parallel machine scheduling with precedence constraints and setup times

Bernat Gacias ^{1,2}, Christian Artigues ^{1,2} and 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

{bgacias,artigues,lopez}@laas.fr

Abstract

This paper presents different methods for solving parallel machine scheduling problems with precedence constraints and setup times between the jobs. Limited discrepancy search methods mixed with local search principles, dominance conditions and specific lower bounds are proposed. The proposed methods are evaluated on a set of randomly generated instances and compared with previous results from the literature and those obtained with an efficient commercial solver. We conclude that our propositions are quite competitive and our results even outperform other approaches in most cases.

Keywords: Parallel machine scheduling, setup times, precedence constraints, limited discrepancy search, local search.

1 Introduction

This paper deals with parallel machine scheduling with precedence constraints and setup times between the execution of jobs. We consider the optimization of two different criteria: the minimization of the sum of completion times and the minimization of maximum lateness. These two criteria are of great interest in production scheduling. The sum of completion times is a criterion that maximizes the production flow and minimizes the work-in-process inventories. Due dates of jobs can be associated to the delivery dates of products. Therefore, the minimization of maximum lateness is a goal of due date satisfaction in order to disturb as less

as possible the customer who is delivered with the longest delay. These problems are strongly *NP-hard* (Graham et al., 1979).

The parallel machine scheduling problem has been widely studied (Cheng and Sin, 1990), specially because it appears as a relaxation of more complex problems like the hybrid flow shop scheduling problem or the RCPSP (Resource-Constrained Project Scheduling Problem). Several methods have been proposed to solve this problem. In Chen and Powell (1999), a column generation strategy is proposed. Pearn et al. (2007) propose a linear program and an efficient heuristic for large-size instances for the resolution of priority constraints and family setup times problem. Salem et al. (2000) solve the problem with a tree search method. More recently, Néron et al. (2008) compare two different branching schemes and several tree search strategies for the problem with release dates and tails for the makespan minimization case.

However, the literature on parallel machine scheduling with precedence constraints and setup times is quite limited. Baev et al. (2002) and van den Akker et al. (2005) deal with the problem with precedence constraints for the minimization of the sum of completion times and maximum lateness respectively. The setup times case is considered in Schutten and Leussink (1996) and in Ovacik and Uzsoy (1995) for the minimization of maximum lateness. Uzsoy and Velasquez (2008) deal with the same criterion on a single machine with family-dependent setup times. Finally, Nessah et al. (2005) propose a lower bound and a branch-and-bound method for the minimization of the sum of completion times.

Problems that have either precedence constraints or setup times, but not both, can be solved by list scheduling algorithms. It means there exists a total ordering of the jobs (i.e., a list) that, when a given machine assignment rule is applied, reaches the optimal solution (Schutten, 1994). For a regular criterion, this rule is called Earliest Completion Time (ECT). It consists in allocating every job to the machine that allows it to be completed at the earliest. This reasoning unfortunately does not work when precedence constraints and setup times are considered together, as shown in Hurink and Knust (2001). We have then to modify the way to solve the problem and consider both scheduling and resource allocation decisions.

In Section 2, we define formally, the parallel machine scheduling problem with setup times and precedence constraints between jobs. In Section 3 we present a branch-and-bound method and its components: tree structure, lower bounds, and dominance rules. Discrepancy-based tree search methods are described in Section 4. In Section 5 we present the hybrid tree-local search methods used to solve large-size instances. Section 6 is dedicated to computational experiments.

2 Problem definition

We consider a set J of n jobs to be processed on m parallel machines. The precedence relations between the jobs and the setup times, considered when different jobs are sequenced on the same machine, must be satisfied. The preemption is not allowed, so each job is continually processed during p_i time units on the same machine. The machine can process no more than one job at a time. The decision variables of the problem are the start times of every job $i = 1..n$, S_i , and let us define C_i as the completion time of job i , where $C_i = S_i + p_i$. Let r_i and d_i be the release date and the due date of job i , respectively. Due dates are only considered for job lateness computation. We denote by E the set of precedence constraints between jobs. The relation $(i, j) \in E$, with $i, j \in J$, means that job i is performed before job j ($i \prec j$) such that job j can start only after the end of job i ($S_j \geq C_i$). Finally, we define s_{ij} as the setup time needed when job j is processed immediately after job i on the same machine. Thus, for two jobs i and j processed successively on the same machine, we have either $S_j \geq C_i + s_{ij}$ if i precedes j , or $S_i \geq C_j + s_{ji}$ if j precedes i . Using the notation of Graham et al. (1979), the problems under consideration are denoted: $Pm|prec, s_{ij}, r_i|\sum C_i$ for the minimization of the sum of completion times and $Pm|prec, s_{ij}, r_i|L_{\max}$ for the minimization of the maximum lateness.

Example

A set of 5 jobs ($n = 5$) must be executed on 2 parallel machines ($m = 2$). For every job i , we give p_i , r_i , d_i , and s_{ij} (see Table 1). Besides, for that example we have the precedence constraints: $1 \prec 4$ and $2 \prec 5$.

(a)				(b)					
n	p_i	r_i	d_i	s_{ij}	1	2	3	4	5
1	4	1	7	1	0	2	3	4	5
2	3	0	5	2	7	0	6	1	3
3	4	3	8	3	2	4	0	7	1
4	3	3	10	4	4	4	8	0	1
5	2	1	5	5	3	4	8	5	0

Table 1: Example 1 data

Figure 1 displays a feasible solution for this problem. The set of precedence constraints is satisfied: $S_5 = 13 \geq 3 = C_2$ and $S_4 = 5 \geq 5 = C_1$. We stress that job 4 must postpone its start time on M_2 by one time unit because of the precedence

constraint. On the other hand, we have to check that, for every job i , $r_i \leq S_i$ and that setup times between two sequenced jobs on the same machine are also respected. For the evaluation of the solution, we observe that for the minimization of the sum of completion times the value of the function is $z = \sum C_i = 43$ and for the minimization of maximum lateness $z = L_{\max} = L_5 = 10$.

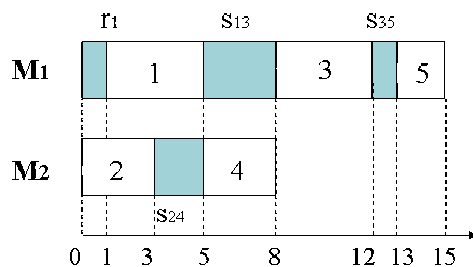


Figure 1: Feasible schedule

3 Branch-and-Bound components for $Pm|prec, s_{ij}, r_i|\sum C_i$ and $Pm|prec, s_{ij}, r_i|L_{\max}$

A tree structure with two levels of decisions (scheduling and resource allocation) is defined in Section 3.1. Lower bounds, constraint propagation mechanisms and dominance rules are introduced in Sections 3.2 and 3.3.

3.1 Tree structure

Precedence constraints and setup times scheduling problems may not be efficiently solved by a list algorithm as conjectured by Hurink and Knust (2001). It means that there possibly does not exist a job allocation rule that reaches an optimal solution where all the possible lists of jobs are enumerated. Let us consider the minimization of the sum of completion times for 4 jobs scheduled on 2 parallel machines. The data of the problem are displayed in Table 2.

(a)		
n	p_i	r_i
1	1	0
2	1	0
3	1	2
4	1	2

(b)				
s_{ij}	1	2	3	4
1	0	10	2	10
2	10	0	1	1
3	10	10	0	10
4	10	10	10	0

Table 2: Example 2 data

If we consider the problem without precedence constraints, we find two optimal solutions ($\sum C_i = 9$) when we allocate the jobs following the Earliest Completion Time rule for the lists $\{1, 2, 4, 3\}$ and $\{2, 1, 4, 3\}$ (see Figure 2a). Now, let us consider the same problem with the precedence constraint $3 \prec 4$. In that case, there does not exist any allocation rule that reaches an optimal solution for any list of jobs that respects the precedence constraint. The optimal solution ($\sum C_i = 11$) is reached when we consider the list $\{1, 2, 3, 4\}$ and job 3 is not allocated on the machine that allows it to finish first (see Figure 2b). Thus, in our problems we have not only to find the best list of jobs but also to specify the best resource allocation.

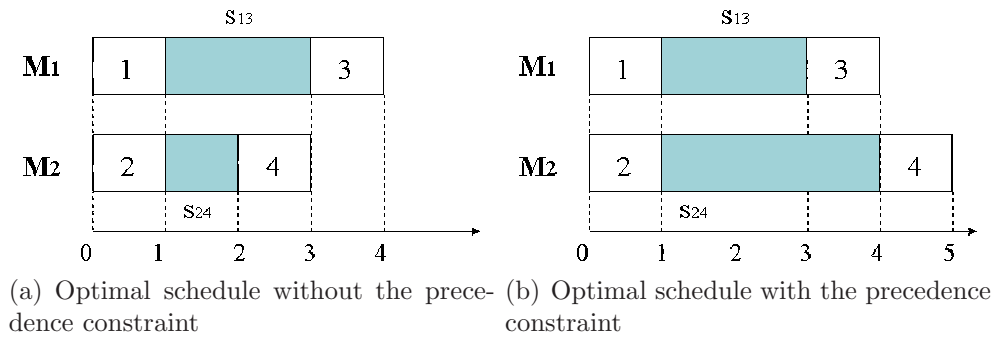


Figure 2: Example of job allocation

The optimal solution can be reached by a two decision-level tree search. We define a node as a partial schedule $\sigma(p)$ of p jobs. Every node entails at most $m \times (n - p)$ child nodes. The term $n - p$ corresponds to the choice of the next job to be scheduled (job scheduling problem). Only the jobs with all the previous jobs already executed are candidates to be scheduled. Once the next job to be scheduled is selected we have to consider the m possible machine allocations (machine allocation problem). For practical purposes, we have mixed both levels of decision: one branch is associated with the choice of the next job to schedule and also with the choice of the machine. A solution is reached when the node represents a complete schedule, that means when $p = n$.

3.2 Node evaluation

Node evaluation differs depending on the studied criterion. First, we propose to compute a simple lower bound. For every node (partial schedule), we update the earliest start times of the unscheduled jobs taking account of the branching decisions through precedence constraints and we calculate the minimum completion time (for $\min \sum C_i$ criterion) and the minimum lateness (for $\min L_{\max}$ criterion) for every not yet-scheduled job. Then we update the criterion and we compare the lower bound with the best current solution.

We propose to compute an upper bound. The upper bound is computed by a simple list scheduling heuristic selecting the combination of job, between the not yet-scheduled jobs, and machine with the shortest start time.

For criterion $\min \sum C_i$, we also propose to compute the lower bound presented in Nessah et al. (2005) for the parallel machine scheduling problem, with sequence-dependent setup times and release dates ($Pm|s_{ij}, r_i|\sum C_i$). This problem is a relaxation of the problem with precedence constraints, so the lower bound is still valid for our problem. In this paper, we just present the lower bound for the problem, that is based on job preemption relaxation, and we refer to Nessah et al. (2005) for the proof.

Let S_* be the schedule obtained with the SRPT (Shortest Remaining Processing Time) rule for the relaxed problem $1|r_i, (\frac{p_i}{m} + s_i^*), pmtn|\sum \max(C_i^* - s_i^*, r_i + p_i)$, where $s_i = \min_{j \neq i} s_{ij}$ and $s_i^* = \frac{s_i}{m}$. Let $C_{[i]}^*(S_*)$ be the modified completion time of job i with the processing time $p_i + s_i^*$ for each job i . Let $a_i = p_i + r_i + s_i^*$ and let $(a_{[1]}, a_{[2]}, \dots, a_{[n]})$ be the series obtained by sorting (a_1, a_2, \dots, a_n) in non-decreasing order. Then $LB = \sum \max[C_{[i]}^*(S_*), a_{[i]}] - \sum s_i^*$ is a lower bound for $Pm|prec, s_{ij}, r_i|\sum C_i$. The complexity of the lower bound is $O(n \log n)$, the same complexity as SRPT.

For $\min L_{\max}$, the evaluation consists in triggering a satisfiability test based on constraint propagation involving energetic reasoning (Lopez and Esquirol, 1996). The energy is produced by the resources and it is consumed by the jobs. We apply

this feasibility test to verify whether the best solution reached from the current node will be at least as good as the best current solution. We determine the minimum energy consumed by the jobs ($E_{consumed}$) over a time interval $\Delta = [t_1, t_2]$ and we compare it with the available energy ($E_{produced} = m \times (t_2 - t_1)$). In our problem we also have to consider the energy consumed by the setup times (E_{setup}). If $E_{consumed} + E_{setup} > E_{produced}$ we can prune the node.

For an interval Δ where there is a set F of k jobs that may consume energy, we can easily show that the minimum quantity of setups which occurs is $\alpha = \max(0, k - m)$. So, we have to take the α shortest setup times of the set $\{s_{ij}\}, i, j \in F$, into account.

The energy consumed in an interval Δ is $E_{consumed} = \sum_i \max(0, \min(p_i, t_2 - t_1, r'_i + p_i - t_1, t_2 - d'_i + p_i)) + \sum_l^\alpha s_{[l]}$ where $s_{[l]}$ are the setup times of the set $\{s_{ij}\}, i, j \in F$, sorted in non-decreasing order, and a time window $[r'_i, d'_i]$ for every not yet-scheduled job i is issued from precedence constraint propagation:

$$r'_i = \max\{r_i, r_j + p_j; \forall j \in \Gamma_i^-\} \text{ and } d'_i = \min\{Z_{best} + d_i, d'_j - p_j; \forall j \in \Gamma_i^+\},$$

where Γ_i^- and Γ_i^+ are respectively the set of previous and successor jobs for job i and Z_{best} is the minimum current value for L_{max} .

In Figure 3 we illustrate how to compute the energy consumed by the not yet-scheduled jobs (1 to 5 in the example) for a 3-machine problem. For every job, we determine a time window and the minimum energy consumed (in grey) over the selected interval $\Delta = [t_1, t_2]$. For E_{setup} we have to take the α shortest setup times, in the example $k = 4$ (there is no consumption for job 1) and $m = 3$, so we have to sum only the shortest setup time between the consuming jobs, in our case we add 2 energy units (value of s_{35}).

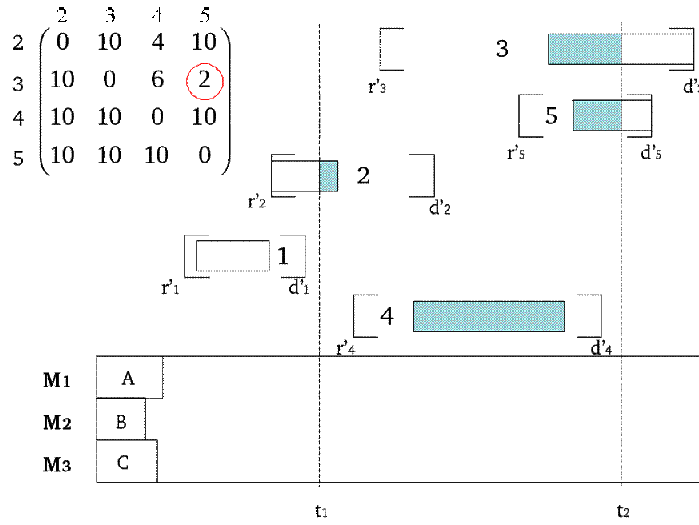


Figure 3: Minimum energy consumed in a partial schedule

The time interval $\Delta = [t_1, t_2]$ considered to compute the energy consumed is $t_1 = \min r'_i, \forall i \in F$ and $t_2 = d'_j$, where j is the job with the shortest time window $\min (d'_j - r'_j), \forall j \in F$. The complexity of the energetic test is $O(n^2)$.

3.3 Dominance rules

We also propose dominance rules to restrict the search space. They consist in trying to find whether there exists a dominant node allowing us to prune the evaluated node. All proposed rules are based on the dominance properties of the set of active schedules. A schedule S is active if no feasible schedule can be obtained from S by left-shifting a single activity. Let us define the *front* of a partial schedule as the set of the last jobs executed on the machines (the ones with the largest start times).

We first present a global dominance rule based on max flow computation based on a resource-flow model previously used for the resource-constrained project scheduling problem with setup times (Neumann et al. (2002), Section 2.13). The idea is to verify that there exists a partial schedule $\sigma'(p)$ with the start times $S' = \{S'_1, S'_2, \dots, S'_i, \dots, S'_p\}$ different from $\sigma(p)$ with start times $S = \{S_1, S_2, \dots, S_i, \dots, S_p\}$ that allows us to move forward the start time of job k without modifying other start times ($S'_i = S_i, \forall i \neq k$ and $S'_k \leq S_k - 1$). This is a necessary but not a sufficient condition for the dominance. Besides, the schedule $\sigma'(p)$ has to keep the same front as $\sigma(p)$ except for the case where job k does not belong to the front of $\sigma'(p)$ (the dominant partial schedule). For example in

Figure 4, job 5 ($S_5 = 18$) may be scheduled after job 4 or between job 2 and job 4 with a shortest start time ($S'_5 = 17$). In the first case the new schedule $\sigma'(p)$ is not dominant because of setup times but in the second case it is, so the front can be modified only if job k is not part of it in $\sigma'(p)$.

We represent $\sigma'(p)$ by a graph and we turn the dominance rule in a max flow computation. Two vertices are considered for every job, the first one represents the start time i_t and the second one the completion time i_s of the job. One unit capacity arcs are defined between the vertices i_s-j_t by the partial schedule $\sigma'(p)$ and they represent the transfer of resource units between the jobs. Finally, we need four dummy vertices. Two vertices $(0_s, 0_t)$, the source node S and the sink node T , flow origin and flow destination, respectively. Arcs $S-0_s$ and 0_t-T have m -unit capacity and represent the resource constraint. 1-unit capacity arcs between $S-i_s$ and i_t-T ensure the job execution.

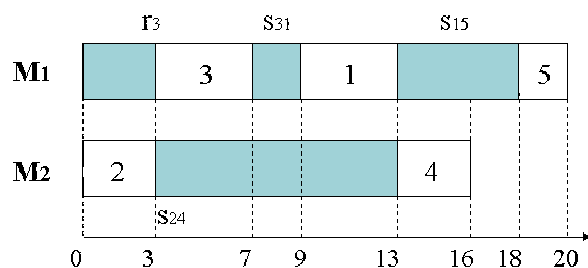


Figure 4: Partial schedule of the evaluated node

Figure 5 shows the flow network for the schedule depicted in Figure 4 (data of Table 1). For each node we try to find a schedule that allows us to move forward the start time of the last scheduled job by one unit (job 5 in the example, $S'_5 = 17$) and to keep the same start times for the other jobs. We create a direct arc i_s-j_t if $S'_j > S'_i + p_i + s_{ij}$, that means if job j can be executed on the same machine than job i . In order to respect the second condition for the dominance, we do not create the arcs between the jobs belonging to the front in the evaluated node (job 4 and job 5). We observe that a max flow of $m + p$ units is necessary to ensure all job executions and to satisfy the resource constraints. In that case, $\sigma'(p)$ is a feasible schedule and we can prune the node.

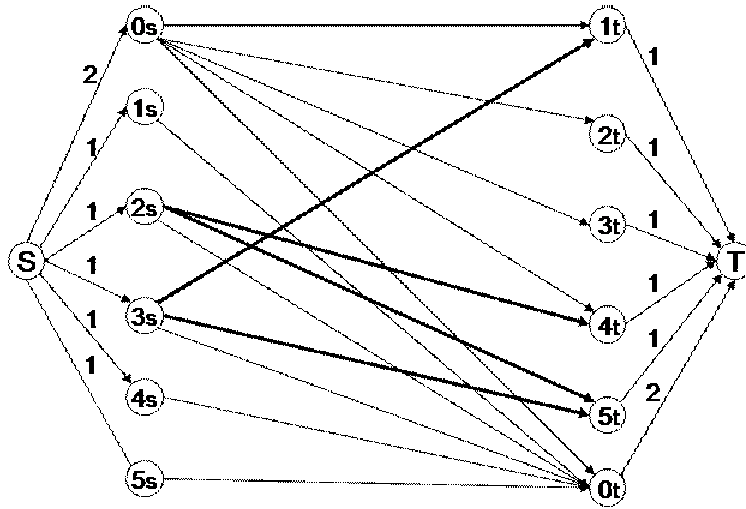


Figure 5: Network to compute the max flow dominance rule

We propose a second dominance rule based on the position of the front jobs in the priority list. For a given schedule, the dominance rule searches for a new list of jobs in order to obtain the dominant partial schedule. We modify the list of scheduled jobs taking into account the precedence constraints. We can prune the evaluated node when the dominant partial schedule keeps the same front than the evaluated node (jobs 1, 2, and 3), one of the jobs starts earlier ($S'_1 < S_1$) and for the rest of jobs belonging to the front the start times are not delayed ($S_2 = S'_2$ and $S_3 = S'_3$), as we see in Figure 6.

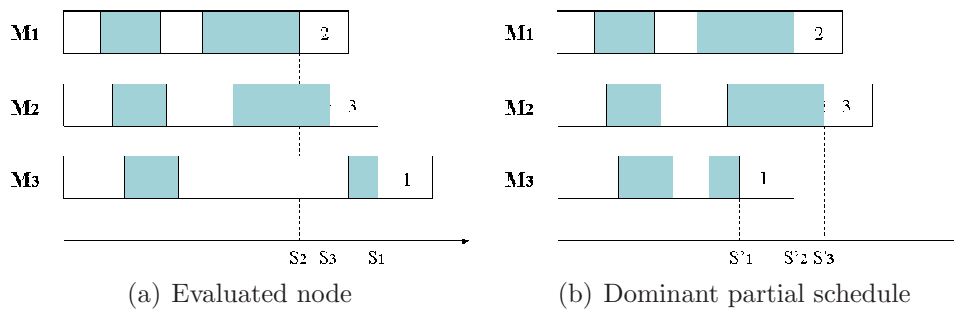


Figure 6: Example of dominant partial schedule

We propose to permute the order of the m front jobs in order to find the dominant schedule. For example, in Figure 6 if the order of scheduled front jobs is 1 – 2 – 3 we test all the possible permutations satisfying precedence constraints.

If one of such permutations yields a dominant partial schedule, we can prune the evaluated node. This rule can be computed with time complexity $O(m!)$. As shown in Section 5, despite its exponential worst-case complexity, this dominance rule has interesting properties when used in conjunction with discrepancy-based tree search and remains efficient for a small number of machines. A partial enumeration remains valid if m becomes very large.

Note similar dominance rules have already been used for the RCPSP (which can be defined as an extension of the parallel machine scheduling problem with precedence constraints, but without setup times) under the name "cutset dominance rules" (Demeulemeester and Herroelen, 1997). However, in Demeulemeester and Herroelen (1997), all the cutsets are kept in memory yielding important memory requirements.

4 Discrepancy-based tree search methods

4.1 Limited discrepancy search

To tackle the combinatorial explosion of the standard branch-and-bound methods for large problem instances, we use a method based on the discrepancies regarding a reference branching heuristic. Such a method is based on the assumed good performance of this reference heuristic, thus making an ordered local search around the solution given by the heuristic. First, it explores the solutions with few discrepancies from the heuristic solution and then it moves away from this solution until it has covered the whole search space. In this context, the principle of *LDS* (*Limited Discrepancy Search*) (Harvey and Ginsberg, 1995) is to explore first the solutions with discrepancies on top of the tree, since it assumes that the early mistakes, where very few decisions have been taken, are the most important.

Figure 7 shows *LDS* behavior for a binary tree search with the number of discrepancies for every node. Let us consider the left branch as the reference heuristic decision. At iteration 0 we explore the heuristic solution, then at iteration 1 we explore all the solutions that differ at most once from the heuristic solution, and we continue until all the leaves have been explored.

LDS can be used as an exact method, for small-size instances, when the maximum number of discrepancies is authorized. We can also use it as an approximate method if we limit the number of authorized discrepancies.

Several methods based on *LDS* have been proposed to improve its efficiency. *ILDS* (*Improved LDS*) (Korf, 1996) has been devised to avoid the redundancy (observed in Figure 7) where the solutions with no discrepancies are also visited at iteration 1. *DDS* (*Depth-bounded Discrepancy Search*) (Walsh, 1997) or *DBDFS* (*Discrepancy-Bounded Depth First Search*) (Beck and Perron, 2000) propose to

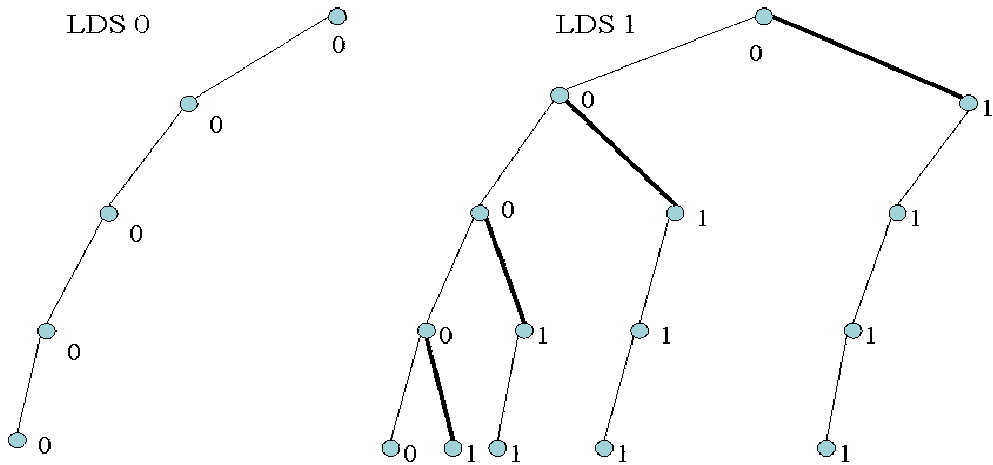


Figure 7: *Limited Discrepancy Search* for a binary tree

change the order of the search. DDS limits the depth where the discrepancies are considered, in the sense that at the k^{th} iteration we only authorize the discrepancies at the first k levels of the tree. It stresses the principle that the early mistakes are the most important. DBDFS consists in a classical *DFS* where the nodes explored are limited by the discrepancies. Recently, in the *YIELDS* method (Karoui et al., 2007), learning process notions are integrated. In what follows, we propose several versions of LDS adapted to the considered parallel machine scheduling context.

4.2 Exploration strategy

As a branching heuristic, we use the same heuristic to compute the lower bound presented in Section 3.2: *EST* (*Earliest Start Time*) rule for the selection of the next job to schedule and the resource to execute it. We take criterion EST because it is intuitively compatible with the minimization of setup times which has globally a positive impact for minimization of other regular criteria (Artigues et al., 2005). In case of tie between two jobs, we apply *SPT* (*Smallest Processing Time*) rule for $\min \sum C_i$ and *EDD* (*Earliest Due Date*) for $\min L_{\max}$.

Because of the existence of two types of decisions, we consider here two types of discrepancies: discrepancy on job selection and discrepancy on resource allocation. In the case of non-binary search trees, we have two different ways to count the discrepancies (see Figure 8). In the first mode (*binary*), we consider that choosing the heuristic decision corresponds to 0 discrepancy, while any other value corresponds to 1 discrepancy. The other mode (*non-binary*) consists in considering that the further we are from the heuristic choice the more discrepancies we have to count. We suggest to evaluate experimentally both modes for the heuristic for job

selection. On the other hand, for the choice of the machine, we use the non-binary mode since we assume that the allocation heuristic only makes a few errors. As we will see in Section 5, selecting the machine which allows the earliest completion of the job is a high performance heuristic.

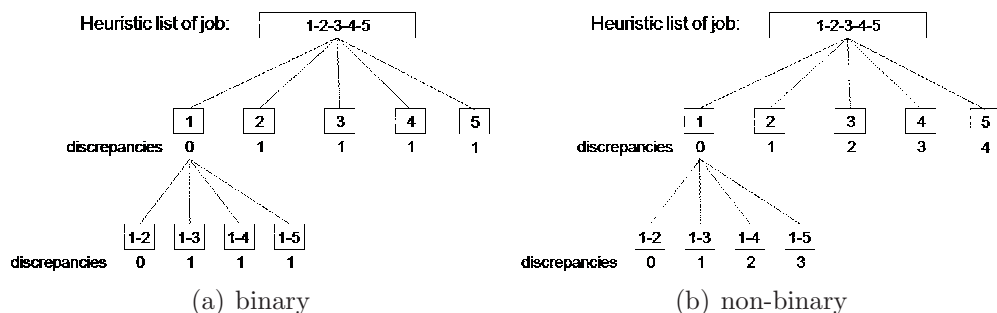


Figure 8: Example of discrepancies counting modes on job selection

We propose to test three different branching schemes. The first one, called DBDFS (Beck and Perron, 2000), is a classical depth-first search where the solutions obtained are limited by the allowed discrepancies (see Section 4.1). We propose two other strategies, *LDS-top* and *LDS-low*, which consider the number of discrepancies for the order in which the solutions are reached. The node to explore is the node with the smallest number of discrepancies, and with the smallest depth for the strategy called *LDS-top*, and with the largest depth for the strategy called *LDS-low*. As Figure 9 shows (case of 2 authorized discrepancies) all three methods explore the same solutions but in different orders.

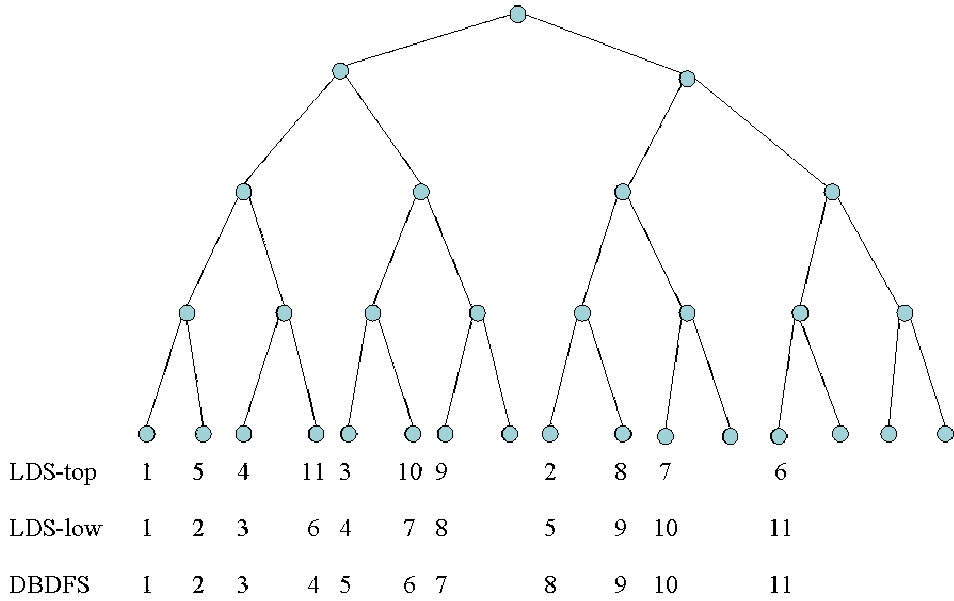


Figure 9: Order of explored leaves for different branching rules

4.3 Large neighborhood search based on LDS

We have presented *LDS* as an exact or a truncated tree search method. In this section, we propose to use it as part of local search. In a local search method, we define a solution neighborhood $N_k(x)$ (k defines the acceptable variations of solution x). If we find a solution x' better than x in $N_k(x)$ then we explore the neighborhood $N_k(x')$ of this new best solution. In the case of large-scale neighborhoods problems, the neighborhood becomes so huge that we can consider the search for the best solution in $N_k(x)$ as an optimization sub-problem (Shaw, 1998). In that context, we consider a neighborhood defined by an *LDS* search tree.

CDS (*Climbing Discrepancy Search*) (Milano and Roli, 2002) is the first large neighborhood search method based on *LDS* (see Algorithm 1). At each iteration it carries out a k -discrepancy search around the best current solution. If a better solution is found, then *CDS* explores its neighborhood. In the case of no better solution

Algorithm 1: Climbing Discrepancy Search

```
begin
   $k \leftarrow 1$ ;
   $k_{max} \leftarrow n$ ;
   $Sol_{ref} \leftarrow InitialHeuristic()$ ;
  while  $k \leq k_{max}$  do
    /* Generate the set of solutions  $N$  of  $k$  discrepancies
        $Sol_{ref}$  */
     $N = LDS(Sol_{ref}, k)$ ;
     $s' \leftarrow BestOf(N)$ ;
    if  $z(s') < z(Sol_{ref})$  then
      |  $Sol_{ref} \leftarrow s'$ ;
      |  $k \leftarrow 1$ ;
    else
      |  $k \leftarrow k + 1$ ;
  end
```

is found, then k is increased by one.

The drawback of CDS is that for large-size instances the neighborhood quickly explodes. Hmida et al. (2007) propose *CDDS* (*Climbing Depth-bounded Discrepancy Search*) that mixes principles of CDS and of DDS. The neighborhood of the best solution is limited not only by the number of discrepancies but also by the depth in the tree. In that case, the neighborhood explosion is avoided and the idea that the most important heuristic mistakes are early ones is stressed.

In this work, we propose two variants of CDS and CDDS for the problems at hand. They are closely related with *VNS* (*Variable Neighborhood Search*) (Hansen and Mladenovic, 2001) concept, since we modify the size and the structure of the neighborhood explored. *HD-CDDS* (*Hybrid Discrepancy CDDS*) (see Algorithm 2) consists in a mix of CDS and CDDS. We start with a CDS search, but if for a defined number of discrepancies k_{limit} we cannot find a better solution, then we authorize a bigger number of discrepancies only between some levels ($[d_{min}, d_{max}]$). Once we have finished the search for $k_{limit} + 1$, we propose either to increase the number of authorized discrepancies and to keep the same number of levels where the discrepancies are authorized ($x = d_{max} - d_{min}$), which is the case in Algorithm 2, or to increase the number of levels and to keep the number of discrepancies. This method solves the problem of neighborhood explosion and offers more jobs mobility than CDDS (which is particularly interesting for setup times problems) but we need to parametrize the values of the search (k_{limit}, x).

Algorithm 2: Algorithm *HD-CDDS*

```
begin
   $k \leftarrow 1$ ;
   $d_{min} \leftarrow 0$ ;
   $d_{max} \leftarrow n$ ;
   $Sol_{ref} \leftarrow InitialHeuristic()$ ;
  while termination conditions not met do
    /* Generate the set of solutions  $N$  of  $k$  discrepancies from
     $Sol_{ref}$  */
     $N = GenSol(Sol_{ref}, k, d_{min}, d_{max})$ ;
     $s' \leftarrow BestOf(N)$ ;
    if  $z(s') < z(Sol_{ref})$  then
       $Sol_{ref} \leftarrow s'$ ;
       $k \leftarrow 1$ ;
       $d_{min} \leftarrow 0$ ;
       $d_{max} \leftarrow n$ ;
    else
      if  $k < k_{limit}$  then
         $k \leftarrow k + 1$ ;
      else
        if  $d_{max} - d_{min} = n$  then
           $d_{min} \leftarrow 0$ ;
           $d_{max} \leftarrow x$ ;
        else
           $d_{min} \leftarrow d_{max}$ ;
           $d_{max} \leftarrow d_{min} + x$ ;
          if  $d_{min} > n$  then
             $k \leftarrow k + 1$ ;
             $d_{min} \leftarrow 0$ ;
             $d_{max} \leftarrow x$ ;
        end
      end
    end
  end
```

The second proposed variant, *MC-CDS (Mix Counting CDS)*, is an application of *CDS* but with a modification in the way to count the discrepancies for the job selection rule only. We consider a binary counting for the discrepancies at the top level of the tree and a non-binary counting way for the rest of levels. This variant accepts discrepancies for all depth levels because the non-binary counting restricts the explored neighborhood.

4.4 Discrepancy-adapted dominance rules

In this section we propose to adapt the second dominance rule presented in Section 3.3 to the principle of local search. We argue that it can be very inefficient to use the dominance rule as presented in Section 3.3 with the proposed local search methods. Indeed, the best solutions of the neighborhood could not be explored because we have found a dominant partial schedule that allows us to prune them. Even if it is true that there exists a solution better than the evaluated node, it may not belong to the explored neighborhood.

For that reason, we propose discrepancy-adapted dominance rules. Once we know the criterion that defines the neighborhood (for example, k authorized discrepancies from the job list L), we only have to verify that the new list of jobs L' that reaches the dominant partial schedule is part of the explored nodes in the local search ($L' \in G$, where G is the set of k -discrepancies lists from L).

We can see that the max flow computation rule presented in Section 3.3 is not discrepancy adaptable. It is not possible to verify that the dominant partial schedule $\sigma'(p)$ is part of the explored space because the rule indicates the existence of $\sigma'(p)$ but not the corresponding schedule. On the other hand, the second dominance rule introduced in Section 3.3 consists in a local modification of the evaluated schedule in order to explicitly obtain the dominant schedule. That way, we have the list of jobs, L' , available to compare with the best current solution list of jobs, L , and to verify that the dominant schedule is part of the explored nodes. Hence, when the encountered dominant schedule is not part of the explored neighborhood the current node is not pruned.

5 Computational experiments

In this section we present the main results obtained from the implementation of our work. In the literature we have not found instances for parallel machines including both setup times and precedence constraints. Therefore, we propose to test the methods on a set of randomly generated instances. The algorithms are implemented in C++ and were run on a 2 GHz personal computer with 2 Go of RAM under the Linux Fedora 8 operating system.

We generate a set of 120 (60 for each criterion) small-size instances ($n = 10$, $m = 3$, and $n = 15$, $m = 2$) for the evaluation of the dominance rules and for the *ECT rule* efficiency. Then, we test on a set of 120 middle-size instances ($n = 40$, $m \in [2, 4]$) the different branching rules (*LDS-top*, *LDS-low*, and *DBDFS*), the different ways to count the discrepancies (*binary* and *non-binary*) to determine the best methods for being included inside the *LDS* structure of the local search methods. The efficiency of the lower bounds, the dominance rules and the ener-

getic reasoning proposed in Section 3 are tested on middle and large-size instances ($n = 100, m \in [2, 4]$). We also compare the *CDS* and the *HD-CDDS* methods with the results obtained in Néron et al. (2008) for the hard instances of the $Pm|r_i, q_i|C_{\max}$ problem (without precedence constraints and setup times). And finally, we evaluate and compare the proposed methods on a set of 120 large-size instances with the results obtained with ILOG OPL 6.0.

We use the RanGen software (Demeulemeester et al., 2003) in order to generate the precedence graph between the jobs. Setup times and time windows $[r_i, d_i]$ cannot be generated by RanGen. Setup times are generated from the uniform distributions $U[1, 10]$ and $U[20, 40]$. Moreover they must respect the weak triangle inequality: $s_{ij} \leq s_{ik} + p_k + s_{kj}, \forall i, j, k$. The values of p_i are generated from the uniform distribution $U[1, 5]$. Time windows are generated in a classical way we found in the literature (Sourd, 2005). The values of d_i are generated from the uniform distribution $U[\max(0, P \times (1 - \tau - \rho/2)), P \times (1 - \tau + \rho/2)]$, where $P = \sum(p_i + \min_j(s_{ij}))$, $\tau \in [0, 1]$, $\rho \in [0, 1]$. The r_i are generated from d_i , $r_i = d_i - (p_i \times (2 + \alpha))$ where $\alpha \in [-0.5, +1.5]$.

We solve to optimality the small-size instances and we compare the results (*Optimal*) with the results obtained when we apply the ECT rule (*ECT*) for each possible list of jobs (jobs are only allocated to the machine which allows to finish it first), with the results using the dominance rule based on the permutation of front jobs (*Front Rule*), and with the results using the dominance rule based on max flow computation (*Max Flow*).

60 Instances			
$n = 10, m = 3$	<i>NbBest</i>	<i>AvgNodes</i>	<i>AvgTCPU</i>
<i>Optimal</i>	60 (100.0 %)	484925	10.6
<i>Front Rule</i>	60 (100.0 %)	480444	12.3
<i>Max Flow</i>	60 (100.0 %)	339541	27.7
<i>ECT</i>	53 (88.3 %)	61684	0.07

60 Instances			
$n = 15, m = 2$	<i>NbBest</i>	<i>AvgNodes</i>	<i>AvgTCPU</i>
<i>Optimal</i>	60 (100.0 %)	10126793	641.9
<i>Front Rule</i>	60 (100.0 %)	9480313	626.4
<i>Max Flow</i>	60 (100.0 %)	7530154	454.6
<i>ECT</i>	54 (90.0 %)	1747416	2.5

Table 3: Results of ECT and dominance rules efficiency for $\min \sum C_i$ problem

60 Instances			
$n = 10, m = 3$	<i>NbBest</i>	<i>AvgNodes</i>	<i>AvgTCPU</i>
<i>Optimal</i>	60 (100.0 %)	281896	5.6
<i>Front Rule</i>	60 (100.0 %)	263474	7.9
<i>Max Flow</i>	60 (100.0 %)	219557	19.7
<i>ECT</i>	52 (86.7 %)	69141	0.07

60 Instances			
$n = 15, m = 2$	<i>NbBest</i>	<i>AvgNodes</i>	<i>AvgTCPU</i>
<i>Optimal</i>	60 (100.0 %)	11936385	884.8
<i>Front Rule</i>	60 (100.0 %)	10503767	778.7
<i>Max Flow</i>	60 (100.0 %)	8945948	628.4
<i>ECT</i>	54 (90.0 %)	4681104	7.27

Table 4: Results of ECT and dominance rules efficiency for min L_{\max} problem

First, note that we found some hard instances that we could not solve to optimality before 15000 seconds. We observe in Tables 3 and 4 that ECT rule is very efficient for both problems. The optimal solution is reached over almost 90 % of the instances and the average CPU time (*AvgTCPU*) is clearly reduced when we use the ECT rule. These results let us consider, for local search methods, only the job permutation allocating the jobs on the machines following the ECT rule. The dominance front rule is also effective, the average number of explored nodes (*AvgNodes*) and the average CPU time usually decrease when we use it. We observe that the *Max Flow* rule largely reduces the number of explored nodes and the CPU time, except for the very small-size instances. We deduce that it is a very efficient rule to solve to optimality instances with a larger number of jobs.

In the comparison between the two different ways to count the discrepancies, *binary* and *non-binary* (only for job selection rules), we have evaluated on the middle-size instances the number of times each mode has found the best solution (*NbBest*). The CPU time is limited to 100 seconds.

Table 5 shows that the binary mode has a higher performance than the non-binary one. Out of a set of 120 instances, the binary mode has found the best solution over 75 % of the instances, independently of the branching rule. We find very similar results for both criteria. In the following, the binary counting is kept for the LDS structure of the local search.

120 Instances $n = 40, m \in [2, 4]$	<i>NbBest</i>	
	<i>binary</i> mode	<i>non-binary</i> mode
<i>DBDFS</i>	90 (75.0 %)	48 (40.0 %)
<i>LDS-top</i>	93 (77.5 %)	49 (40.8 %)
<i>LDS-low</i>	98 (81.7 %)	31 (25.8 %)

Table 5: Results of the comparison between discrepancies counting modes

In Table 6, we can see the results for the comparison between the exploration strategies. In addition to previous notations, we introduce the average mean deviation from the best solution (*AvgDev*). The CPU time is limited to 100 seconds.

Binary mode $n = 40, m \in [2, 4]$	$\min \sum C_i$ (60 instances)		$\min L_{\max}$ (60 instances)	
	<i>NbBest</i>	<i>AvgDev</i>	<i>NbBest</i>	<i>AvgDev</i>
<i>DBDFS</i>	43 (71.7 %)	0.91 %	47 (78.3 %)	1.86 %
<i>LDS-top</i>	29 (48.3 %)	0.43 %	17 (28.3 %)	2.33 %
<i>LDS-low</i>	50 (83.3 %)	0.71 %	59 (98.3 %)	0.75 %

Table 6: Results for the comparison of different branching strategies

We find that LDS-low is the most efficient strategy, since it reaches the best solution for a larger number of instances and it presents the less important average mean deviation when the best solution is found by another strategy. LDS-low finds the best solution for all instances except for one corresponding to the maximum lateness minimization and for 50 over a set of 60 instances for completion times sum minimization. We use this strategy for the remaining computational experiments.

The lower bounds, the energetic reasoning, and the discrepancy-adapted dominance rule are compared in Tables 7 and 8. We run a 30 seconds LDS search for the middle and large-size instances for different versions of the node evaluation. First, we only consider the lower bound computed using precedence constraint propagation (*LBCP*), then we add the lower bound (*LB_{NCY}*) proposed in Nesah et al. (2005) for $\min \sum C_i$ problem and the energetic reasoning (*ENERGY*) for $\min L_{\max}$ problem; finally we add the discrepancy-adapted dominance rule (*DaDR*). We compare the number of times each version finds the best solution (*NbBest*), the explored nodes average (*AvgNodes*), and the average CPU time needed to reach the best solution (*TBest*), only for the cases that all versions have found it.

60 Instances			
$n = 40, m \in [2, 4]$	<i>NbBest</i>	<i>AvgNodes</i>	<i>TBest</i>
<i>LBCP</i>	36 (60.0 %)	62007	4.52
<i>LB_{NCY}</i>	38 (63.3 %)	61742	4.47
<i>DaDR</i>	35 (58.3 %)	53373	1.69

60 Instances			
$n = 100, m \in [2, 4]$	<i>NbBest</i>	<i>AvgNodes</i>	<i>TBest</i>
<i>LBCP</i>	26 (43.3 %)	9259	17.55
<i>LB_{NCY}</i>	34 (56.7 %)	7813	15.63
<i>DaDR</i>	38 (63.3 %)	7606	8.71

Table 7: Results of lower bounds and dominance rule efficiency for $\min \sum C_i$ problem

Tables 7 and 8 show the efficiency of the specific lower bound LB_{LCY} and energetic reasoning with the computation of setup times consumption. Moreover, we find that the discrepancy-adapted dominance rule is very efficient for large-size instances but not especially interesting for the middle-size instances. However the time consumed to reach the best solution is reduced when we use the dominance rule for most of cases.

60 Instances			
$n = 40, m \in [2, 4]$	<i>NbBest</i>	<i>AvgNodes</i>	<i>TBest</i>
<i>LBCP</i>	47 (78.3 %)	93737	4.81
<i>ENERGY</i>	48 (80.0 %)	99856	4.24
<i>DaDR</i>	44 (73.3 %)	71737	4.59

60 Instances			
$n = 100, m \in [2, 4]$	<i>NbBest</i>	<i>AvgNodes</i>	<i>TBest</i>
<i>LBCP</i>	44 (73.3 %)	11474	4.29
<i>ENERGY</i>	48 (80.0 %)	12961	3.58
<i>DaDR</i>	55 (91.7 %)	9462	3.17

Table 8: Results of lower bound, energetic reasoning and dominance rule efficiency for $\min L_{\max}$ problem

We compare *CDS* and *HD-CDDS* methods against other tree search methods presented in Néron et al. (2008). In Néron et al. (2008), the authors test two

different branching schemes, time windows (tw) and chronological (chr), and several incomplete tree search techniques (truncated branch-and-bound, LDS, Beam Search and Branch-and-Greed) for the $Pm|r_i, q_i|C_{\max}$ problem. We adapt the proposed methods for this problem and we use the heuristic for the initial solution and the upper bounds proposed in their paper. In Table 9, we compare LDS (z is the number of authorized discrepancies) and Beam Search (BS , ω is the number of explored child nodes) results, the method with the best results in their work, against the proposed methods CDS and HD-CDDS. We have evaluated the number of times the method has found the best solution ($NbBest$) and for how many of them the method is the only one to reach the best solution ($NbBestStrict$) for a set of 50 hard instances ($n = 100$ and $m = 10$). The CPU time is limited to 30 seconds as in Néron et al. (2008).

50 instances	$NbBest$	$NbBestStrict$
$LDS_{z=1}^{tw}$	1 (2.0 %)	0
$LDS_{z=2}^{chr}$	7 (14.0 %)	0
$BS_{\omega=3}^{tw}$	25 (50.0 %)	3
$BS_{\omega=4}^{chr}$	22 (44.0 %)	0
<i>CDS</i>	35 (70.0 %)	6
<i>HD-CDDS</i>	38 (76.0 %)	9

Table 9: Results for the comparison with other truncated tree search techniques

Although precedence constraints and setup times are not considered in the problem, we can observe that our propositions are strictly better. Out of a set of 50 instances, CDS and HD-CDDS find the best solution for most of the cases and they find a new best solution for 6 and 9 instances respectively. Rather than contradicting the statement of relative LDS inefficiency for parallel machine problem experienced by Néron et al. (2008), this demonstrates, at least for this problem, the efficiency of large neighborhood search based on LDS.

Finally, we compare the local search methods with the results obtained by ILOG OPL 6.0. The four variants of the hybrid tree local search methods (*CDS*, *CDDS*, *HD-CDDS*, *MC-CDS*) are implemented with *LDS-low*, discrepancy-adapted dominance rule and binary counting (except for *MC-CDS* which supposes a mix counting). We solve the large-size instances ($n = 100, m \in [2, 4]$) for two different CPU time limits, 30 and 300 seconds, then we compare the number of times when the best solution has been found by the method and the average deviation from the best solution.

30 instances $p \sim U[1, 5], s_{ij} \sim U[1, 10]$	$TCPU = 30s$		$TCPU = 300s$	
	<i>NbBest</i>	<i>AvgDev</i>	<i>NbBest</i>	<i>AvgDev</i>
<i>CDS</i>	17 (56.6 %)	0.64 %	7 (23.3 %)	0.51 %
<i>CDDS</i>	7 (23.3 %)	0.75 %	7 (23.3 %)	0.82 %
<i>HD-CDDS</i>	16 (53.3 %)	0.60 %	14 (46.7 %)	0.43 %
<i>MC-CDS</i>	17 (56.6 %)	0.64 %	10 (33.3 %)	0.45 %
<i>ILOG OPL</i>	4 (13.3 %)	1.51 %	2 (6.7 %)	1.47 %

30 instances $p \sim U[1, 5], s_{ij} \sim U[20, 40]$	$TCPU = 30s$		$TCPU = 300s$	
	<i>NbBest</i>	<i>AvgDev</i>	<i>NbBest</i>	<i>AvgDev</i>
<i>CDS</i>	9 (30.0 %)	0.23 %	6 (20.0 %)	0.18 %
<i>CDDS</i>	7 (23.3 %)	0.35 %	6 (20.0 %)	0.38 %
<i>HD-CDDS</i>	12 (40.0 %)	0.26 %	11 (36.6 %)	0.17 %
<i>MC-CDS</i>	11 (36.7 %)	0.25 %	13 (43.3 %)	0.26 %
<i>ILOG OPL</i>	10 (33.3 %)	0.70 %	5 (16.6 %)	0.63 %

Table 10: Results for the comparison of different variants of hybrid tree local search methods for $\min \sum C_i$ problem

In Table 10, we observe that hybrid local search methods improve the best solutions found by ILOG OPL. All methods, except CDDS, find the best solution for a large number of instances and the mean deviation from the best solution are less important than ILOG OPL solutions. We observe that computing an upper bound highly increases the efficiency of the truncated search.

30 instances $p \sim U[1, 5], s_{ij} \sim U[1, 10]$	TCPU = 30s		TCPU = 300s	
	<i>NbBest</i>	<i>AvgDev</i>	<i>NbBest</i>	<i>AvgDev</i>
<i>CDS</i>	10 (33.3 %)	2.75 %	7 (23.3 %)	3.06 %
<i>CDDS</i>	9 (30.0 %)	2.65 %	8 (26.7 %)	3.28 %
<i>HD-CDDS</i>	13 (43.3 %)	1.92 %	10 (33.3 %)	2.56 %
<i>MC-CDS</i>	13 (43.3 %)	1.75 %	11 (30.0 %)	2.29 %
<i>ILOG OPL</i>	15 (50.0 %)	2.07 %	18 (60.0 %)	1.55 %

30 instances $p \sim U[1, 5], s_{ij} \sim U[20, 40]$	TCPU = 30s		TCPU = 300s	
	<i>NbBest</i>	<i>AvgDev</i>	<i>NbBest</i>	<i>AvgDev</i>
<i>CDS</i>	3 (10.0 %)	2.76 %	2 (6.0 %)	2.89 %
<i>CDDS</i>	3 (10.0 %)	2.71 %	2 (6.0 %)	2.88 %
<i>HD-CDDS</i>	13 (43.3 %)	2.12 %	7 (23.3 %)	1.55 %
<i>MC-CDS</i>	12 (40.0 %)	2.08 %	8 (26.7 %)	1.83 %
<i>ILOG OPL</i>	15 (50.0 %)	0.91 %	19 (63.3 %)	0.90 %

Table 11: Results for the comparison of different variants of hybrid tree local search methods for min L_{\max} problem

Table 11 shows the results for the minimization of maximum lateness. For this case, we observe ILOG OPL improves our results, but we can say that the proposed methods are still competitive, the mean deviation is acceptable and they found the best solution over 50 % and 37 % of instances, for 30 and 300 seconds respectively.

6 Conclusion

In this paper we have studied limited discrepancy-based search methods. We have compared and tested some of the existing options for different LDS components, such as discrepancy counting modes and branching structures, to solve the parallel machine scheduling problem with precedence constraints and setup times.

New local search methods based on LDS have been proposed and compared with similar existing methods. The computational experiments show these methods are efficient to solve parallel machine scheduling problems in general and demonstrates the interest, at least for the studied problem, of incorporating LDS into a large neighborhood search scheme as first suggested by Milano and Roli (2002).

We have suggested an energetic reasoning scheme integrating setup times and we have proposed new global and local dominance rules adapted to discrepancies.

As the results show, these evaluation techniques allow to reduce the number of explored nodes and the time of the search.

As a direction for further research, the proposed methods could be extended to solve more complex problems involving setup times, like the hybrid flow shop or the RCPSP.

References

- C. Artigues, P. Lopez, and P-D. Ayache. Schedule generation schemes and priority rules for the job-shop problem with sequence-dependent setup times: dominance properties and computational analysis. *Annals of Operations Research*, 138(1): 21–52, 2005.
- I. D. Baev, W. M. Meleis, and A. Eichenberger. An experimental study of algorithms for weighted completion time scheduling. *Algorithmica*, 22:34–51, 2002.
- J. C. Beck and L. Perron. Discrepancy-bounded depth first search. In *Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CP-AI-OR'00)*, Paderborn (Germany), 2000.
- Z.-L. Chen and W. B. Powell. Solving parallel machine scheduling problems by column generation. *INFORMS J. on Computing*, 11(1):78–94, 1999.
- T. Cheng and C. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1990.
- E. Demeulemeester, M. Vanhoucke, and W. Herroelen. Rangen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6:17–38, 2003.
- E. L. Demeulemeester and W. S. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11): 1485–1492, 1997.
- R.L Graham, E.L Lawler, J.K Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, pages 287–326, 1979.
- P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
- W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of 14th IJCAI*, 1995.

- A. Ben Hmida, M. J. Huguet, P. Lopez, and M. Haouari. Climbing depth-bounded discrepancy search for solving hybrid flow shop scheduling problems. *European Journal of Industrial Engineering*, 1(2):223–243, 2007.
- J. Hurink and S. Knust. List scheduling in a parallel machine environment with precedence constraints and setup times. *Operations Research Letters*, 29:231–239, 2001.
- W. Karoui, M.-J. Huguet, P. Lopez, and W. Naanaa. YIELDS: A yet improved limited discrepancy search for CSPs. In *4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, Brussels (Belgium), 2007.
- R. Korf. Improved limited discrepancy search. In *Proceedings of 13th AAAI*, 1996.
- P. Lopez and P. Esquirol. Consistency enforcing in scheduling: A general formulation based on energetic reasoning. In *5th International Workshop on Project Management and Scheduling (PMS'96)*, Poznan (Poland), 1996.
- M. Milano and A. Roli. On the relation between complete and incomplete search: an informal discussion. In *Proceedings CPAIOR'02*, Le Croisic (France), 2002.
- E. Néron, F. Tercinet, and F. Sourd. Search tree based approaches for parallel machine scheduling. *Computers and Operations Research*, 35(4):1127–1137, 2008.
- R. Nessah, Ch. Chu, and F. Yalaoui. An exact method for $Pm|sds, r_i| \sum C_i$ problem. *Computers and Operations Research*, 34:2840–2848, 2005.
- K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources*. Springer, 2002.
- I. M. Ovacik and R. Uzsoy. Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International journal of production research*, 33(11):3173–3192, 1995.
- W. L. Pearn, S. H. Chung, and C .M. Lai. Scheduling integrated circuit assembly operations on die bonder. *IEEE Transactions on electronics packaging manufacturing*, 30(2), 2007.
- A. Salem, G. C. Anagnostopoulos, and G. Rabadi. A branch-and-bound algorithm for parallel machine scheduling problems. In *Society for Computer Simulation International (SCS)*, Portofino (Italy), 2000.
- J. M. J. Schutten. List scheduling revisited. *Operations Research Letters*, 18:167–170, 1994.

- J. M. J. Schutten and R. A. M. Leussink. Parallel machine scheduling with release dates, due dates and family setup times. *International journal of production economics*, 46-47(1):119–126, 1996.
- P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming-CP 98*, 1998.
- F. Sourd. Earliness-tardiness scheduling with setup considerations. *Computers and Operations Research*, 32(7):1849–1865, 2005.
- R. Uzsoy and J. D. Velasquez. Heuristics for minimizing maximum lateness on a single machine with family-dependent set-up times. *Computers and Operations Research*, 35:2018–2033, 2008.
- M. van den Akker, J. Hoogeveen, and J. Kempen. Parallel machine scheduling through column generation: minimax objective functions, release dates, deadlines and/or generalized precedence constraints. Technical report, Utrecht university, 2005.
- T. Walsh. Depth-bounded discrepancy search. *APES Group, Department of Computer Science*, 1997.