A branch and bound method for the job-shop problem with sequence-dependent setup times

Christian Artigues, Dominique Feillet

A branch and bound method for the job-shop problem with sequence-dependent setup times

Christian Artigues[1] and Dominique Feillet[2]

[1] LAAS–CNRS
7 avenue du Colonel Roche, 31077 Toulouse, France
artigues@laas.fr

[2]LIA – Université d'Avignon
339 chemin meinajariés, Agroparc, BP 1228, 84911 Avignon Cedex 9, France
Dominique.Feillet@univ-avignon.fr

Abstract

This paper deals with the job-shop scheduling problem with sequence-dependent setup times. We propose a new method to solve the makespan minimization problem to optimality. The method is based on iterative solving via branch and bound decisional versions of the problem. At each node of the branch and bound tree, constraint propagation algorithms adapted to setup times are performed for domain filtering and feasibility check. Relaxations based on the traveling salesman problem with time windows are also solved to perform additional pruning. The traveling salesman problem is formulated as an elementary shortest path problem with resource constraints and solved through dynamic programming. This method allows to close previously unsolved benchmark instances of the literature and also provides new lower and upper bounds.

keywords: job-shop scheduling, sequence-dependent setup times, branch and bound, constraint propagation, dynamic programming

1 Introduction

This work deals with the job-shop problem with sequence dependent setup times (SDST-JSP). The job-shop problem considers the scheduling of a set

of jobs on distinct machines. This problem is widely investigated in the literature and many efficient approaches exist for its resolution [10, 32, 25, 43]. The SDST-JSP is a variant problem where machines have to be reconfigured between two consecutive operations. We address here the optimal solution of the SDST-JSP with a makespan minimization criterion. The method is based on the iterative resolution of decisional versions of the problem, via branch and bound. Within branch and bound trees, constraint propagation algorithms are performed for domain filtering and feasibility check. Relaxations based on the traveling salesman problem with time window (TSPTW) are also solved to perform additional pruning. The traveling salesman problem is formulated as an elementary shortest path with resource constraints [22] and solved through dynamic programming.

We present the considered problem in Section 2 and the traveling salesman problem with time windows relaxation in Section 3. The disjunctive graph representation is recalled in Section 4. A review of the relevant literature is given in Section 5. The sketch of the branch and bound algorithm is provided in Section 6 and its components are detailed in the subsequent Sections. The heuristic used to compute feasible solutions is described in Section 7. Feasibility checks and domain filtering algorithms based on constraint propagation are described in Section 8. Section 9 gives the formulation of the TSPTW as an elementary shortest path problem with resource constraints and presents the dynamic programming algorithm used to solve it. In Section 10, the proposed branching scheme and the associated dominance rule are described. In Section 11, the results of the branch and bound method on the set of instances proposed by Brucker and Thiele [13] are discussed. Concluding remarks are drawn in Section 12.

2 The job-shop problem with sequence dependent setup times

The SDST-JSP considers a set of $n \geq 1$ jobs $\mathcal{J} = \{J_i\}_{1 \leq i \leq n}$ and a set of $m \geq 1$ machines $\mathcal{M} = \{M_k\}_{1 \leq k \leq m}$. Each job $J_i \in \mathcal{J}$ is defined as a set of m operations $J_i = \{O_{ij}\}_{1 \leq j \leq m}$. Each operation O_{ij} has a non zero integer duration $p_{ij} \in \mathbb{N}$ and requires a single machine $m_{ij} \in \mathcal{M}$. The operations of a same job all require distinct machines and are subject to precedence constraints. Operation O_{ij} precedes operation $O_{i,j+1}$ for all $i = 1, \ldots, n$ and for all $j = 1, \ldots, m-1$. The set of all operations is denoted $\mathcal{O} = \cup_{1 \leq i \leq n} J_i$ while $\mathcal{O}_k$ denotes the set of operations assigned to machine M_k.

A sequence dependent setup time, denoted s_{ijk}, is defined for each couple

of distinct jobs (J_i, J_j) and for each machine M_k. An initial setup time s_{0ik} is defined for each job J_i and for each machine M_k. It is assumed that the triangular inequality holds: for each machine M_k and for each triplet of distinct jobs (J_i, J_j, J_x), we have $s_{ixk} \leq s_{ijk} + s_{jxk}$; for each machine M_k and for each couple of distinct jobs (J_i, J_j), we have $s_{0jk} \leq s_{0ik} + s_{ijk}$.

A schedule is a mapping $\mathcal{T}$ of operations to time periods where $\mathcal{T} = (t_{ij})_{O_{ij} \in \mathcal{O}}$ and $t_{ij} \in \mathbb{N}$. In this paper we consider the makespan objective for the SDST-JSP. This particular problem is denoted $J|s_{ij}|C_{\max}$ in the standard three-field scheduling notation and we will denote it (P) in the remaining of the paper.

$$(P) \quad \min C_{\max} \tag{1}$$

subject to

$$C_{\max} \geq t_{ij} + p_{ij} \qquad \forall O_{ij} \in \mathcal{O} \tag{2}$$

$$t_{ij} + p_{ij} \leq t_{i,j+1} \qquad \forall J_i \in \mathcal{J}, \forall j \in [1, m-1] \tag{3}$$

$$t_{ij} + p_{ij} + s_{ixk} \leq t_{xy} \textbf{ or}$$

$$t_{xy} + p_{xy} + s_{xik} \leq t_{ij} \qquad \forall O_{ij}, O_{xy} \in \mathcal{O}_k, O_{ij} \neq O_{xy} \tag{4}$$

$$t_{ij} \geq s_{0im_{ij}} \qquad \forall O_{ij} \in \mathcal{O} \tag{5}$$

The objective (1) is the minimization of the makespan $C_{\max}$, the maximum completion time of all jobs (2). Constraints (3) are the job precedence constraints, stating that an operation $O_{i,j+1}$ cannot start before the end of its preceding operation O_{ij} in job J_i. It follows that the operations of a job form a chain. Constraints (4) are the machine constraints which state that two distinct operations O_{ij} and O_{xy} sharing the same machine k cannot be scheduled simultaneously and that the machine must be set up between two consecutive operations. Hence either O_{ij} cannot start before the end of O_{xy} plus the necessary setup time s_{xik}, or O_{xy} cannot start before the end of O_{ij} plus the necessary setup time s_{ixk}. Constraints (5) enforce the start time of any operation to occur after the initial setup time on its assigned machine.

An alternative representation of the setup times lies in defining a set $\mathcal{F}$ of $f \geq 1$ families such that each operation $O_{ij} \in \mathcal{O}$ is associated with a family $f_{ij} \in \mathcal{F}$. Now setup time s_{ixk} on machine M_k from an operation $O_{ij} \in \mathcal{O}_k$ of job J_i to an operation $O_{xy} \in \mathcal{O}_k$ of job J_x can be written $s_{f_{ij} f_{xy}}$. This representation can be interesting when the number of families is significantly smaller than the number of operations. Then, efficient preprocessing techniques can be carried out (see Section 8.1).

The job-shop scheduling problem with sequence dependent setup times is a NP-hard problem, as an extension of the standard job-shop scheduling

problem, denoted $J||C_{\max}$, where all setup times are equal to zero. In this paper, the optimal solution of (P) is searched by iteratively solving the decisional version (FP) defined as follows. Let $T \geq 0$ be a tentative upper bound of the makespan. We consider problem $FP(T)$ as the problem of finding a schedule $\mathcal{T}$ such that $C_{max} \leq T$ and constraints (2-5) are satisfied. Let UB denote the makespan of any feasible schedule. Let LB denote a lower bound of the optimal solution of (P). Let $C^*_{\max}$ denote the optimal solution of (P). We have:

$$C^*_{\max} = \min_{LB \leq T \leq UB} \{T | FP(T) \text{ has a solution}\} \tag{6}$$

3 The traveling salesman problem with time windows relaxation

When $m = 1$, problem (P) is denoted $1|s_{ij}|C_{max}$ and is equivalent to the traveling salesman problem (TSP) as stated by Conway et al [20]. Since $m = 1$ we drop the machine and operation indices in the notations. We can define a depot represented by node 0 and n cities represented by nodes $1, \ldots, n$. Then the distance matrix $(l_{ij})_{0 \leq i,j \leq n}$ is given by

$$l_{0i} = s_{0i} \qquad \forall i \in [1, n] \tag{7}$$

$$l_{ij} = s_{ij} \qquad \forall i, j \in [1, n], i \neq j \tag{8}$$

$$l_{i0} = 0 \qquad \forall i \in [0, n] \tag{9}$$

Since the sum of the operation processing times is a constant, problem $1|s_{ij}|C_{max}$ and the above defined TSP are equivalent. It follows that when $m > 1$, a lower bound LB_{TSP} of the optimal solution of (P) can be computed as follows by using $l^*(\text{TSP}_{\mathcal{O}_k})$, the length of the optimal tour of the traveling salesman relaxation considering only the operations on M_k.

$$LB_{\text{TSP}} = \max_{M_k \in \mathcal{M}} (l^*(\text{TSP}_{\mathcal{O}_k}) + \sum_{O_{ij} \in \mathcal{O}_k} p_{ij}) \tag{10}$$

This relaxation can be strengthened by computing time windows for all operations given an upper bound UB of the optimal makespan. Let r_{ij} denote the earliest start time of operation O_{ij}. Let d_{ij} denote the latest completion time of operation O_{ij}. For a given upper bound UB we can

compute the following valid values for the time windows:

$$r_{i1} = s_{0im_{i1}} \qquad \forall J_i \in \mathcal{J} \tag{11}$$

$$r_{ij} = \max(s_{0im_{ij}}, r_{i(j-1)} + p_{i(j-1)}) \qquad \forall J_i \in \mathcal{J}, \forall j \in [2, m] \tag{12}$$

$$d_{im} = UB \qquad \forall J_i \in \mathcal{J} \tag{13}$$

$$d_{ij} = d_{i(j+1)} - p_{i(j+1)} \qquad \forall J_i \in \mathcal{J}, \forall j \in [1, m-1] \tag{14}$$

Any solution of (P) with a makespan not greater than UB must obviously verify $r_{ij} \leq t_{ij} \leq d_{ij} - p_{ij}$ for each operation $O_{ij} \in \mathcal{O}$. In Section 8, we will present methods to compute tighter valid time windows. Let us now consider the relaxation of (P) considering only operations assigned to machine k. We obtain a makespan minimization one machine problem with time windows and sequence-dependent setup time denoted $1|r_i, d_i, s_{ij}|C_{max}$, the solution of which is a lower bound for (P). When setup times are all equal to 0, the problem is already NP-hard but it can be efficiently solved by the well-known Carlier's algorithm [16]. Otherwise this problem is a travelling salesman problem with time windows (TSPTW) with depot 0, cities $1, \ldots, n$, distance matrix $(l_{ij})_{0 \leq i,j \leq n}$ given by (7-9), service times p_i for $1 \leq i \leq n$ and time windows $[r_i, d_i - p_i]$. More precisely the problem is a TSPTW variant where the objective function is the minimization of total travel times plus total waiting time [24, 3, 8]. Let $C^*_{\max}(\text{TSPTW}_{\mathcal{O}_k})$ denote the optimal solution of the TSPTW relaxation considering only the operations on M_k. We obtain the lower bound LB_{TSPTW} as follows.

$$LB_{\text{TSPTW}} = \max_{M_k \in \mathcal{M}} (C^*_{\max}(\text{TSPTW}_{\mathcal{O}_k})) \tag{15}$$

Note that this lower bound depends on the upper bound UB and on the way the time windows are computed.

Let us now consider the decisional version $FP(T)$ tackled in this paper. Since we have a tentative upper bound T and a feasibility problem, the subproblem reduces to the search feasible solutions for the above-defined TSPTW. Finding a machine k such that the associated $\text{TSPTW}_{\mathcal{O}_k}$ is infeasible would indeed mean than value T renders $FP(T)$ infeasible. This problem (denoted F-TSPTW in what follows) is known to be NP hard [39]. In the next Section, we will see how comparable relaxations are tackled by the previously proposed exact methods for the SDST-JSP and its variants. In this paper, we propose to solve the TSPTW relaxation of problem $FP(T)$ exactly by a dynamic programming algorithm (see Section 9), the time windows being sharpened by constraint propagation techniques (see Section 8).

We use the solution of the relaxations to prune the search in the case unfeasibility is proven or to derive a feasible solution to (P) when a feasible TSPTW solution is found.

4 The disjunctive-graph representation

A useful tool for the solution of scheduling problems is the so-called disjunctive graph initially proposed for the standard job-shop problem [38]. This graph provides an efficient representation of the decisions, while limiting the solution space. It is defined as follows.

Let $G = (X, U, E)$ be the disjunctive graph. The set of vertices X is made up of the set of operations plus two dummy vertices representing the beginning and the end of the schedule. Thus, X has $n \times m + 2$ vertices. A set of arcs U and a set of edges E are defined. Arcs in U represent precedence constraints between operations and are called conjunctive arcs. They are weighted with the processing time of the origin vertex of the arc. Edges in E represent disjunctive constraints between operations on a same machine and are called disjunctive arcs. Actually, disjunctive arcs can be interpreted as the union of two exclusive arcs with opposite directions.

By definition, a selection is a state of the disjunctive graph where a direction is chosen for some disjunctive arcs. A selection is said to be complete when every arc has a direction. A complete selection corresponds to a unique semi-active schedule (in which no operation can be left-shifted of one time unit) if the resulting graph is acyclic. Once they are directed, disjunctive arcs are weighted with the sum of the processing time of the origin vertex of the arc plus the setup time required between the origin and the destination vertices. Minimizing the makespan then reduces to the search of the longest path in the graph, the makespan being the length of such a path. Hence, (P) can be defined as the problem of finding a complete acyclic selection for which the longest path is minimum. This standpoint relies on the property that it is possible to consider only semi-active scheduling, (when disjunctive and conjunctive constraints are satisfied), to find an optimal solution.

5 Literature review

Problem $J|s_{ij}|C_{\max}$ received only little attention in the literature, albeit being a natural extension of the job-shop problem with many applications in manufacturing [2]. In this Section, we review the methods previously proposed to solve this problem and its close variants or extensions that

we classify into three categories: priority-rule based (dispatching) methods, local search methods and exact methods.

5.1 Priority-rule based methods

The earliest proposed methods are based on simulation: each time a machine becomes available the dispatching rule selects a job already waiting for this machine, i.e. which as no predecessor or whose predecessor is completed before the machine gets available. Wilbrecht and Prescott [45] were to our knowledge the first to study under this framework the influence of setup times on job shop performance. They propose a simple but efficient priority rule which aims at selecting the job yielding the smallest setup time. Kim and Bobrowski [26] extend the preceding work to various performance measures and propose compound priority rules. Noivo and Ramalhinho-Lourenço [31] propose also new prority-rules considering setup times and compare them with classical priority-rules. Ovacik and Uzsoy [35] propose a more sophisticated priority-rule based algorithm for problem $J|s_{ij}|L_{\max}$ with reentrant jobs (a job may use a machine more than once). Indeed the set of candidate operations considered when a machine becomes available is no more restricted to the already pending operations but extended to the operations having their preceding operation in process. This is justified intuitively because they may require a sufficiently large setup time to be taken in consideration. Theoretical aspects of the interest of extending this candidate set is linked to dominance properties of active schedules (see Section 7 and Artigues et al [5]). Ovacik and Uzsoy were to the best of our knowledge the first to use an exact solution of the TSPTW relaxation as a routine to guide the search. Indeed, among the proposed priority-rules, one amounts to select the first operation in the optimal sequence of the one-machine subproblem restricted to the set of candidate operations. This subproblem is here denoted $1|r_i, s_{ij}|L_{\max}$ and its decisional variant amounts to F-TSPTW. This problem is possibly solved by a specific branch and bound method [36] which has reasonable computational times only for less than 10 operations.

As a priority-rule heuristic embedded in their exact method that will be discussed later, Brucker and Thiele [13] propose an extension of the Giffler-Thompson algorithm which defines an active schedule generation scheme in the standard job-shop problem. They propose a second heuristic based on maximal matchings on bipartite graph. Brucker and Thiele [13] also generated a set of 15 SDST-JSP instances which we use in Section 11 to evaluate our method and compare our results with the state-of-the-art exact methods.

Artigues et al [5] propose several new priority-rule based methods, improving the results obtained by Brucker and Thiele [13] on the 10 largest BT instances. One of these methods being used in our exact procedure, we will give more details in Section 7 and explain why it has better dominance properties than all the previously-proposed priority-rule based methods.

5.2 Local search methods

Among local search methods, Choi and Korkmaz [18] propose a mixed-integer programming formulation of the problem. They formally identify the problem as a multi-traveling salesman problem with side constraints (precedence constraints). They propose a heuristic based upon sequentially identifying a pair of operations that provides a minimum lower bound on the makespan of the associated two-job/m-machine problem with release times. They show their heuristic outperform the one proposed by Zhou and Egbelu [46]. Choi and Choi [19] propose a local search method based on the iterative application of dispatching rules for a problem involving alternative operations in conjunction with setup-times.

The other local search methods proposed so-far for the SDST-JSP are generally based on the disjunctive graph representation. Based on this model, Candido et al [15] propose a genetic algorithm for a job-shop problem involving setup times among multiple practical characteristics such as multiresource operations, machine calendars and alternative routings. A simple tabu search heuristic has been tested on the BT instances by Artigues and Buscaylet [14]. This latter method has been applied by Artigues et al [4] to improve the results on the 10 largest BT instances of the priority-rule based methods proposed by Artigues et al [5].

A large number of local search methods are based on the shifting bottleneck heuristic. This efficient heuristic has been proposed by Adams et al [1] for the job shop problems. It has been successfully extended to tackle practical characteristics of actual job shops. Focusing only on extension to setup times, several variants of the shifting bottleneck heuristic have been proposed [34, 40, 41, 30, 8]. The shifting bottleneck heuristic can be defined as a large neighborhood search which solves at each iteration exactly or approximately the one-machine relaxation of the problem. Hence, most of these methods solve in a specific way the TSPTW relaxation and are therefore of interest for our study. However because of the complexity of the subproblem and because most of these works describe applications on large-size industrial problems, the sub-problem is mostly solved by heuristics [34, 40, 41, 30]. The subproblem considered by Balas et al [8] corresponds

to the TSPTW subproblem which provides the LB_{TSPTW} lower bound described in Section 3. To solve this subproblem, the authors transform the TSPTW in a TSP with precedence constraints having a special structure. This special structure allows to solve the TSP in linear time by dynamic programming [6, 7]. However the TSPTW subproblem does not have in general the required restrictions and the methods is turned into an efficient heuristic. The shifting bottleneck heuristic proposed by Balas et al [8] obtains equivalent or better solutions than the multistart tabu search method of Artigues et al [4] for all the BT instances.

5.3 Exact methods

To our knowledge, the only exact methods that have been developped so far for the SDST-JSP are the ones of Brucker and Thiele [13], Focacci et al [23] and Artigues et al [3].

Brucker and Thiele [13] propose a branch and bound method that extends the one proposed in Brucker et al [12] for the standard JSP. The problem considered by Brucker and Thiele [13] is the general shop problem with sequence-dependent setup time which includes the SDST-JSP as a special case. The method is based on the disjunctive graph representation and a node of the branch and bound tree corresponds to a partial selection. Lower bounds are computed at each node by applying several polynomially-solvable extentions of the Jackson's preemptive schedule relaxation [17] to setup times. Branching is based on the analysis of blocks of operations on the critical path of a feasible schedule issued from the current node. The method was able to solve to optimality the 5 smallest BT instances and provided lower and upper bounds for the 10 largest instances.

At each node, Brucker and Thiele [13] apply filtering algorithms to compute the earliest start times r_{ij} of the operations, so-called heads, as well as the tails q_{ij} which are linked to the latest start times by the relation $q_{ij} = UB - d_{ij}$ and also to deduce implied precedence constraints that enrich the current selection. These filtering algorithms are based on the so-called immediate selections and edge-finding techniques that have been shown to be very efficient to solve the standard JSP [17, 11, 9]. These techniques can be defined as constraint propagation algorithms for the one-machine problem. To extend efficiently these algorithms to the presence of setup times, Brucker and Thiele propose several lower bound of the minimal setup time necessary between operations sharing the same machine.

Vilím and Barták [44] proposed another version of the one-machine constraint propagation algorithms, in the context of a batching and scheduling

problem with sequence-dependent setup times. In Section 8.2, we describe the simple version of the filtering algorithm we have implemented for the branch and bound method proposed in this paper, adapting the algorithms of Brucker and Thiele [13], Vilím and Barták [44] and Nuijten [33] to the SDST-JSP.

Focacci et al [23] propose a branch-and bound method in a constraint programming framework for another variant of the SDST-JSP involving alternative resources for the operations. They also consider total setup time minimization in conjunction with makespan minimization. Their method combines the standard (with no setup) one-machine constraint propagation algorithms and reduced cost-based domain filtering techniques. The latter is based on the relaxation of the TSPTW into an assignment problem solvable in polynomial time. Solving the relaxation provides a lower bound and also a reduced cost matrix. Such reduced costs give information about the additional increase of the lower bound when the corresponding variable assignement is performed. Whenever the increase exceeds the upper bound then the value can be removed from the variable domain. It is worth to note that the reduced cost-based constraint propagation methods have also been sucessfully applied to solve TSPTW instances [24]. For the SDST-JSP, Focacci et al [23] do not report results for the makespan minimization only. However, when the objective is to minimize the makespan and the total setup time in a lexicographic way, they found all the optimal makespans of the 5 smallest BT instances. They also report upper bounds for 2 large instances.

A preliminary version of the branch and bound procedure proposed in this paper was presented by Artigues et al [3]. The main idea of the method is to solve exactly the F-TSPTW relaxation of problem (FP) to prune the search instead of solving the Jackson's preemptive schedule relaxation as in [13] or the assignment problem relaxation as in [23]. (FP) is formulated as a one-machine scheduling problem and solved by means of the commercial solver (ILOG scheduler). In addition, the TSPTW solutions found during the search are stored in a dictionnary. Each time a TSPTW problem has to be solved, the dictionnary is searched to avoid the subproblem resolution if a feasible solution is encountered. This method solved all the smallest BT instances to optimality and closed 2 open instances.

6 The branch and bound method

The branch and bound method we propose is based on the disjunctive graph representation. It aims at making further experiments in the direction initiated by Artigues et al [3]. As in the method of Brucker and Thiele [13], a node of the branch and bound tree corresponds to a partial selection $\mathcal{E}$ where $\mathcal{E}$ is a set of directed arcs obtained from a partial or complete orientation of disjunctive edges E. A node is also defined by operation time windows compatible with the current selection. Hence a node ν is denoted by a triple $(\mathcal{R}, \mathcal{D}, \mathcal{E})$ with $\mathcal{R} = (r_{ij})_{O_{ij} \in \mathcal{O}}$ and $\mathcal{D} = (d_{ij})_{O_{ij} \in \mathcal{O}}$. The branch and bound is given in Algorithm 1. It takes as input a tentative makespan T and returns a failure or a success and in the latter case a feasible schedule $\mathcal{T}$. The time windows are initialized according to T and equations (11-14) (step 1).The root node with an empty selection is created (step 2). A first constraint propagation algorithm (SHAVING) is applied to check the feasibility of the root node and to strenghen the time windows and insert arcs in the selection (step 3). The algorithm is described in Section 8.3. If no unfeasibility is detected by SHAVING, the branch and bound algorithm starts and a node stack is initialized with the root node (step 6). Steps 8 to 19 are repeated until the stack is empty or a maximal number of nodes is reached. For each node ν, a second (faster) constraint propagation algorithm (PROPAGATE) is applied (step 9). This algorithm is detailed in Section 8.2. If unfeasibility is not detected, PROPAGATE has possibly tightened the time windows and the selection of the current node, and a dynamic programming method SOLVETSPTW (described in Section 9) is used to solve the F-TSPTW relaxation (step 10). If SOLVETSPTW does not detect unfeasibility, a solution (represented by start time $\mathcal{T}'$) to the m TSPTW has been found. A heuristic is then applied (step 11). This heuristic, which searches a solution from the current node ν information and solution $\mathcal{T}'$ of the TSPTW, is described in Section 7. If solution $\mathcal{T}$ found by the heuristic has a makespan not greater than T, the problem is feasible and the branch and bound stops. Otherwise, we have to update the node stack by branching (step 16). The branching procedure is described in Section 10.

7 A priority-rule based serial heuristic

The function HEURISTIC returns a feasible schedule $\mathcal{T}$ taking as input the current node $\nu = (\mathcal{R}, \mathcal{D}, \mathcal{E})$ and a TSPTW solution $\mathcal{T}'$ possibly computed. Note that $\mathcal{T}'$ statisfies in any case the machine constraints but possibly

Algorithm 1 BranchAndBound$(T, \mathcal{T})$: returns failure or success

1: initialize $\mathcal{R}^0$ and $\mathcal{D}^0$ with (11-14) and $UB = T$
2: $\nu^0 \leftarrow (\mathcal{R}^0, \mathcal{D}^0, \emptyset)$
3: **if** Shaving(ν^0) =failure **then**
4: **Solution Found**: unfeasible
5: **else**
6: $BBQ \leftarrow \{\nu^0\}$
7: **while** BBQ is not empty **do**
8: $\nu \leftarrow$pop(BBQ)
9: **if** Propagate$(\nu) \neq$failure **then**
10: **if** SolveTSPTW$(\mathcal{T}')(\nu) \neq$failure **then**
11: $\mathcal{T} \leftarrow$Heuristic$(\nu, \mathcal{T}')$
12: **if** $C_{\max}(\mathcal{T}) \leq T$ **then**
13: **return** success
14: **else**
15: $BBQ \leftarrow$Branch(ν, BBQ)
16: **end if**
17: **end if**
18: **end if**
19: **end while**
20: **return** failure
21: **end if**

violates one or more precededence constraints.

HEURISTIC first check whether setting $\mathcal{T} = \mathcal{R}$, $\mathcal{T} = \mathcal{D} - p$ or $\mathcal{T} = \mathcal{T}'$ yields a feasible schedule. In that case, this solution is returned. Otherwise we apply a priority-rule based heuristic SERIALSGS which returns a feasible schedule taking a priority function $\pi : \mathcal{O} \rightarrow \mathbb{R}$.

SERIALSGS is applied 3 times by setting successively $\pi = \mathcal{R}$, $\pi = \mathcal{D} - p$, $\pi = \mathcal{T}'$ and the schedule with the lower makespan returned. We explain herafter the principles and the interest of SERIALSGS.

None of the other priority-rule based heuristics described in Section 5.1 was explicitely designed for having the ability to generate (all) schedules in the classical active, semi-active and non-delay schedule sets. However, it is well-known that active and semi-active schedules are dominant for makespan generation. Artigues et al [5] propose several schedule generation schemes aiming at generating semi-active, active and non-delay schedules and study the dominance properties of the reachable schedules. This analysis reveals that most of the priority-rule based heuristics proposed so far generate sets of schedules possibly excluding all optimal solutions. For instance, the extension of the Giffler and Thompson algorithm proposed by Brucker and Thiele [13] does not generate dominant schedules. In this paper we use the serial schedule generation scheme (SGS) which generates only active schedules and for which the dominance property holds [5]. This algorithm is a is an adaptation for the SDST-JSP of the serial SGS widely used for the resource-constrained project scheduling problem [27]. Algorithm 2 implements the serial schedule generation scheme with an $O(nm^2)$ time complexity.

8 Feasibility tests and adjustments based on constraint propagation

8.1 Setup preprocessing

The feasibility tests we have implemented make use of extensions to setup times of well-known one-machine constraint propagation algorithms, such as immediate selections and edge-finding [17, 11, 9]. Such extensions have been proposed by Brucker and Thiele [13] and Vilím and Barták [44]. Immediate selections aim at detecting new precedence constraints among operations assigned to the same machine, and consequently enrich the current selection by new conjunctive arcs. To perform these deductions the constraint propagation algorithms make use of the minimal duration without idle times of subsets of operations $\Omega \subseteq \mathcal{O}_k$, for each machine $M_k \in \mathcal{M}$. When there are

Algorithm 2 SERIALSGS(π): returns a feasible schedule $\mathcal{T}$

$Q \leftarrow \{O_{i1} | i = 1, \ldots, n\}$
$S_k \leftarrow \emptyset$, $\forall k \in \mathcal{M}$
while Q is not empty **do**
  select $O_{i^*j^*} \leftarrow \arg\min_{O_{ij} \in Q} \pi(O_{ij})$ and let $k = m_{i^*j^*}$.
  $t \leftarrow s_{0i^*k}$
  for each operations $O_{xy} \in S_k$ **do**
    if $t + p_{i^*j^*} + s_{i^*xk} \leq t_{xy}$ **then**
      insert $O_{i^*j^*}$ before O_{xy} in S_k
      break
    else
      $t \leftarrow \max(t, t_{xy} + p_{xy} + s_{xi^*k})$
    end if
  end for
  if $O_{i^*j^*} \notin S_k$ **then**
    append $O_{i^*j^*}$ to S_k
  end if
  $t_{i^*j^*} \leftarrow t$
  $Q \leftarrow Q \setminus \{O_{i^*j^*}\}$
  if $j^* < m$ **then**
    $Q \leftarrow Q \cup \{O_{i^*(j^*+1)}\}$
  end if
end while
return $\mathcal{T} = (t_{ij})_{O_{ij} \in \mathcal{O}}$

no setup times, the minimal duration of a set Ω is denoted p_Ω and is trivially equal to $p_\Omega = \sum_{O_{ij} \in \Omega} p_{ij}$.

When there are setup times, the minimal duration of $\Omega \subseteq \mathcal{O}_k$, is equal to $p_\Omega + s_\Omega$ where s_Ω is the solution of a modified TSP_Ω where the distance from the depot to each node is null. Hence, given a set Ω, computing s_Ω is NP-hard.

When the setup times are computed with operations families, operations of a same family can be gathered together and sequenced consecutively. Hence, the TSP might equivalently be solved with a single operation of each family $f \in \mathcal{F}_\Omega = \cup_{O_{ij} \in \Omega} \{f_{ij}\}$. If $|\mathcal{F}_\Omega|$ is sufficiently lower than $|\Omega|$ then the computational time of the corresponding TSP may become tractable. In the following we note equivalently $s_{\mathcal{F}_\Omega}$ and s_Ω.

The filtering algorithm needs more precisely, for each family subset $\mathcal{F}' \subseteq \mathcal{F}_{\mathcal{O}_k}$ and for each family $f \in \mathcal{F}'$, the values denoted $s_{f \to \mathcal{F}'}$ and $s_{\mathcal{F}' \to f}$. These values correspond to the minimal setup times to schedule a set of operations yielding family set $\mathcal{F}'$ if f is the family of the first (last) operation of the set, respectively. Note that we have $s_{\mathcal{F}'} = \min_{f \in \mathcal{F}'} s_{f \to \mathcal{F}'} = \min_{f \in \mathcal{F}'} s_{\mathcal{F}' \to f}$.

Brucker and Thiele [13] use several TSP lower bounds to estimate $s_{f \to \mathcal{F}'}$ and $s_{\mathcal{F}' \to f}$ and use also the optimal value obtained by complete enumeration, with an $O(|\mathcal{F}'|!)$ time complexity.

In this paper we follow the method of Vilím and Barták [44] who propose to precompute values $s_{f \to \mathcal{F}'}$ and $s_{\mathcal{F}' \to f}$ once before starting the search. Each famility set $\mathcal{F}'$ is coded by a binary representation of the corresponding permutation so that $s_{f \to \mathcal{F}'}$ and $s_{\mathcal{F}' \to f}$ are obtainable in $O(1)$ once the precomputation is made. Values $s_{f \to \mathcal{F}'}$ for each subset $\mathcal{F}' \subseteq \mathcal{F}_{\mathcal{O}_k}$ and each family $f \in \mathcal{F}'$ can be computed in $O(|\mathcal{F}_{\mathcal{O}_k}|^2 2^{|\mathcal{F}_{\mathcal{O}_k}|})$ by the following recursion:

$$s_{f \to \{f\}} = 0 \qquad \forall F_f \subseteq \mathcal{F}_{\mathcal{O}_k} \tag{16}$$

$$s_{f \to \mathcal{F}' \cup \{f\}} = \min_{g \in \mathcal{F}'} \{s_{fg} + s_{g \to \mathcal{F}'}\} \qquad \forall \mathcal{F}' \subseteq \mathcal{F}_{\mathcal{O}_k}, \forall f \in \mathcal{F}_{\mathcal{O}_k} \setminus \mathcal{F}' \tag{17}$$

Values $s_{\mathcal{F}' \to f}$ for each subset $\mathcal{F}' \subseteq \mathcal{F}_{\mathcal{O}_k}$ and each family $f \in \mathcal{F}'$ can be computed in a symetric fashion.

We also introduce the following notation linked to a set of operations Ω useful in what follows.

$$r_\Omega = \min_{O_{ij} \in \Omega} r_{ij} \tag{18}$$

$$d_\Omega = \max_{O_{ij} \in \Omega} d_{ij} \tag{19}$$

8.2 Local constraint propagation

The function PROPAGATE($\mathcal{R}, \mathcal{D}, \mathcal{E}$) of our branch and bound procedure aims at computing adjustments on earliest start times $\mathcal{R}$ and on latest completion times $\mathcal{D}$ compatible with the current selection $\mathcal{E}$, returning a failure when unfeasibility is detected.

It calls iteratively functions UPDATEEARLIESTSTART, UPDATELATEST-COMPLETION, IMMEDIATESELECTION and EDGEFINDING, each performing specific deductions. The sequence of calls is repeated until a failure occurs or no more adjustments are performed. Algorithms UPDATEEARLIESTSTART and UPDATELATESTCOMPLETION are adapted from Brucker and Thiele [13]. They update r_{ij} and d_{ij} for all operations $O_{ij} \in \mathcal{O}$ considering only the precedence relations given by the job precedence constraints and current selection $\mathcal{E}$. The earliest start times can be updated with the following rules :

$$r_{ij} \geq s_{0im_{ij}} \qquad \forall O_{ij} \in \mathcal{O} \tag{20}$$

$$r_{ij} \geq r_{i(j-1)} + p_{i(j-1)} \qquad \forall O_{ij} \in \mathcal{O}, j > 1 \tag{21}$$

$$r_{ij} \geq r_\Omega + p_\Omega + s_{\mathcal{F}_{\Omega\cup\{f_{ij}\}} \to f_{ij}} \qquad \forall O_{ij} \in \mathcal{O}, \; \forall \Omega \subseteq \{O_{xy} | (O_{xy}, O_{ij}) \in \mathcal{E}\} \tag{22}$$

The latest completion times can be updated with the following symetric rules :

$$d_{ij} \leq T \tag{23}$$

$$d_{ij} \leq d_{i(j+1)} - p_{i(j+1)} \qquad \forall O_{ij} \in \mathcal{O}, j < m \tag{24}$$

$$d_{ij} \leq d_\Omega - p_\Omega - s_{f_{ij} \to \mathcal{F}_{\Omega\cup\{f_{ij}\}}} \qquad \forall O_{ij} \in \mathcal{O}, \; \forall \Omega \subseteq \{O_{xy} | (O_{xy}, O_{ij}) \in \mathcal{E}\} \tag{25}$$

Algorithm 3 describes UPDATEEARLIESTSTART. Its time complexity is $O(n^2 m \log n)$. Brucker and Thiele [13] prove that the algorithm provides the largest adjustment according to rule (22) provided that the triangular inequality is verified for setup times. Note that the complexity of our algorithm is lower than the one obtained by Brucker and Thiele [13] thanks to the setup preprocessing. Since the procedures are applied iteratively the consistency check perfomed in step 12 detects any cycle in $G(\mathcal{E}) = (X, U \cup \mathcal{E})$. Algorithm UPDATELATESTCOMPLETION is symetric.

Algorithm IMMEDIATESELECTION is a simple constraint propagation algorithm for the disjunctive resource constraints whose objective is to detect

Algorithm 3 updateEarliestStart($\mathcal{R}, \mathcal{D}, \mathcal{E}$): returns failure or success

for $O_{ij} \in \mathcal{O}$ **do**
  $r_{ij} \leftarrow \max(r_{ij}, s_{0im_{ij}})$
  if $j > 1$ **then**
    $r_{ij} \leftarrow \max(r_{ij}, r_{i(j-1)} + p_{i(j-1)})$
  end if
  $\Omega \leftarrow \{O_{xy} | (O_{xy}, O_{ij}) \in \mathcal{E}\}$
  sort operations of Ω according to increasing earliest start times
  for $O_{xy} \in \Omega$ **do**
    $r_{ij} \leftarrow \max(r_{ij}, p_\Omega + s_{\mathcal{F}_\Omega \cup \{f_{ij}\} \rightarrow f_{ij}})$
    $\Omega \leftarrow \Omega \setminus \{O_{xy}\}$
  end for
  if $r_{ij} + p_{ij} > d_{ij}$ **then**
    return failure
  end if
end for
return success

new precedence constraints between two operations assigned to the same machine. It is based on the rules: $\forall M_k \in \mathcal{M}, \forall O_{ij}, O_{x,y} \in \mathcal{O}_k, O_{ij} \neq O_{x,y}$:

$$r_{ij} + p_{ij} + p_{xy} + s_{ixk} > d_{xy} \implies (O_{xy}, O_{ij}) \in \mathcal{E} \tag{26}$$

$$(O_{xy}, O_{ij}) \in \mathcal{E} \implies r_{ij} \geq r_{xy} + p_{xy} + s_{xik} \text{ and } d_{xy} \leq d_{ij} - p_{ij} - s_{xik} \tag{27}$$

It is implemented trivially in $\mathcal{O}(n^2 m)$. immediateSelection returns a failure whenever any update on the time window of an operation O_{ij} yields $r_{ij} + p_{ij} > d_{ij}$.

Last, algorithm edgeFinding aims at performing additional immediate selections and time windows adjustments considering an operation O_{ij} assigned to a machine M_k and a set $\Omega \subseteq \mathcal{O}_k \setminus \{O_{ij}\}$. More precisely it is based on the following so-called primal and dual rules. The primal rule states that $\forall M_k \in \mathcal{M}$, $\forall O_{ij} \in \mathcal{O}_k$, $\forall \Omega \subseteq \mathcal{O}_k \setminus \{O_{ij}\}$:

$$r_{\Omega \cup \{O_{ij}\}} + p_{\Omega \cup \{O_{ij}\}} + s_{\mathcal{F}_{\Omega \cup \{O_{ij}\}}} > d_\Omega \implies \forall O_{xy} \in \Omega, (O_{xy}, O_{ij}) \in \mathcal{E} \tag{28}$$

The dual rule states that $\forall M_k \in \mathcal{M}$, $\forall O_{ij} \in \mathcal{O}_k$, $\forall \Omega \subseteq \mathcal{O}_k \setminus \{O_{ij}\}$

$$d_{\Omega \cup \{O_{ij}\}} - p_{\Omega \cup \{O_{ij}\}} - s_{\mathcal{F}_{\Omega \cup \{O_{ij}\}}} < r_\Omega \implies \forall O_{xy} \in \Omega, (O_{ij}, O_{xy}) \in \mathcal{E} \tag{29}$$

Roughly both primal and dual rules detect that operation O_{ij} is not insertable inside set Ω. The primal rule detects in addition that O_{ij} has to be scheduled after all the operations of Ω on machine M_k whereas the dual rule detects that O_{ij} has to be scheduled before all the operations of Ω. Rules (22) and (25) can be combined with the primal and dual rules, respectively, to perform the time windows adjustments brought by the new arcs added in $\mathcal{E}$.

The combination of the primal and dual rule gives the following feasiblity check, also called consistency rule. It generates a failure whenever the following condition holds. $\forall M_k \in \mathcal{M}$, $\forall \Omega \subseteq \mathcal{O}_k$

$$r_\Omega + p_\Omega + s_{\mathcal{F}_\Omega} > d_\Omega \tag{30}$$

Vilím and Barták [44] propose a $O(|\mathcal{F}|n^2)$ algorithm to perfom all edge finding deductions and an other $O(n \log n)$ algorithm to perform all deductions of the consistency rule. In this paper we propose an extension of Nuijten's edge finding algoritm [33] for which all consistency checks and adjustments are performed jointly. This extension, which runs in $O(|\mathcal{F}|n^2)$, is detailed in Algorithm 4 for the primal part.

We recall the principles of Nuijten's algorithm and explain the changes we have made to take account of setup times, in the spirit of the method proposed by Vilím and Barták [44].

The main loop enumerate all possible operation families f. The second level loop makes all the updates relative to all relevant sets Ω such that $d_\Omega = d_{ij}$. The first third-level loop (steps 6-17), plays too different roles. Through the update of C, it performs the consistency checks (30). Since setup times verify the triangular inequality, set $\Omega(O_{xy}, O_{ij}) = \{O_{vw} \in \mathcal{O}_k | d_\Omega \leq d_{ij}, r_\Omega \geq r_{xy}\}$ is dominant for rule (30) over any other set Ω' such that $r_{\Omega'} = r_{xy}$ and $r_{\Omega'} = d_{ij}$. The set of sets Ω generated at step 9 includes all sets $\Omega(O_{xy}, O_{ij})$. Hence C provides the largest possible left side value of inequality (30). The second role is to prepare, through the computation of D, the updates of the earliest start times of family f operations due to rules (28) and (22). Hence at step 16, c_{xy} is the largest earliest start time update of an operation O_{xy} if O_{xy} has to be scheduled after all previously generated relevant sets Ω, such that $r_\Omega \geq r_{xy}$. Note that this update is only valid here if $f_{xy} = f$. The second third-level loop (steps 6-17) makes the adjustements detected by rules (28) and (22). The same relevant sets Ω are generated in the reverse order, starting from the last generated one. At step 21, H corresponds to the largest earliest completion time of the all previously generated sets Ω such that $r_\Omega \leq r_{xy}$ provided that a type f operation will be inserted in

Algorithm 4 PRIMALEDGEFINDING$(k, \mathcal{R}, \mathcal{D})$: returns failure or success

for $f \in \mathcal{F}_{\mathcal{O}_k}$ **do**
  for $O_{ij} \in \mathcal{O}_k$ according to increasing earliest start times **do**
    $P \leftarrow 0$
    $C \leftarrow 0$
    $\Omega \leftarrow \emptyset$
    for $O_{xy} \in \mathcal{O}_k$ according to decreasing earliest start times **do**
      if $d_{xy} \leq d_{ij}$ **then**
        $P \leftarrow P + p_{xy}$
        $\Omega \leftarrow \Omega \cup \{O_{xy}\}$
        $C \leftarrow \max(C, r_{xy} + P + s_{\mathcal{F}_\Omega})$
        $D \leftarrow \max(D, r_{xy} + P + s_{\mathcal{F}_\Omega \cup \{f\} \rightarrow f})$
        if $C > d_{ij}$ **then**
          return failure
        end if
      end if
      $c_{xy} \leftarrow D$
    end for
    $H \leftarrow 0$
    for $O_{xy} \in \mathcal{O}_k$ according to increasing earliest start times **do**
      if $d_{xy} \leq d_{ij}$ **then**
        $H \leftarrow \max(H, r_{xy} + P + s_{\mathcal{F}_\Omega \cup \{f\}})$
        $P \leftarrow P - p_{xy}$
        $\Omega \leftarrow \Omega \setminus \{O_{xy}\}$
      else if $f_{xy} = f$ **then**
        if $r_{xy} + P + p_{xy} + s_{\mathcal{F}_\Omega \cup \{f\}} > d_{ij}$ **then**
          $r_{xy} \leftarrow \max(r_{xy}, c_{xy})$
        end if
        if $H + p_{xy} > d_{ij}$ **then**
          $r_{xy} \leftarrow \max(r_{xy}, D)$
        end if
      end if
    end for
  end for
end for
return success

the set, considering only the setup time of this operation (the left side of the inequality of rule (28) minus the operation duration which is here still unknown). Any operation O_{xy} such that $d_{xy} > d_{ij}$ is candidate for being updated by rule (28). Step 25 performs the test of rule (28) for such an operation O_{xy} and current set Ω, verifying $r_\Omega \geq r_{xy}$. Note that the current set is stronger than all the sets that remains to be generated for operation O_{xy} and rule (28), since the setup times verify the triangular inequality and $P = p_\Omega$ is maximal. Step 26 updates the earliest start time if the test is positive using value c_{xy} (see definition above). Step 28 performs the test of rule (28) for operation O_{xy} and the strongest set previously generated, and verifying $r_\Omega \leq r_{xy}$ (see computation of H above). Step 29 performs the update of the earliest start time using value D (see definition above), the largest earliest completion time among all sets Ω such that $d_\Omega = d_{ij}$, (because no operation of any of these sets can start after O_{xy}).

8.3 Global constraint propagation

At the root node of the branch and bound tree, we perform a global constraint propagation algorithm, so-called SHAVING, which can be seen as a one level breadth first search from the root node where the possible start times of all operations inside their time windows are tried, yielding possibly $\sum_{O_{ij} \in \mathcal{O}} (d_{ij} - p_{ij} - r_{ij})$ tries. More precisely for each oeration O_{ij}, a tentative to reduce its time window to $[r_{ij}, r_{ij}]$ is done and function PROPAGATE is called. If a failure occurs then r_{ij} is set to $r_{ij} + 1$ and the process iterates until a global failure occurs (the time window becomes empty) or there are no more adjustments. If no global failure occurs, another operation is considered. The same process is carried out for the latest start time. At the end of the SHAVING process, either a global failure is detected, or the time windows are "shaved". We refer to [37, 29, 42, 21] for more efficient implementations and other variants of the shaving technique.

9 Feasibility tests based on TSPTW solutions

The TSPTW can be defined as a particular elementary shortest path problem with resource constraints (ESPPRC) [22]. The ESPPRC considers a network, a origin node, a destination node and arcs valuated by a cost c_{uv} and consuming resources. The number of resources is denoted Q. Traversing an arc (u, v) consumes an amount l^q_{uv} of each resource q, with $1 \leq q \leq Q$. Values l^q_{uv} are assumed to satisfy the triangle inequality for each resource. Each node u of the network is associated with an interval $[a^q_u, b^q_u]$ such that

the consumption of a resource q along a path from the origin node to u is contrained to belong to the interval. More precisely, if the elementary path uses arc (u, v), the consumption W_v^q of resource q between the origin node and v has to satisty $W_v^q \geq \max(a_v^q, W_u^q + l_{uv}^q)$ and $W_v^q \leq b_v^q$. The objective is to find an elementary path of minimal cost from the origin to the destination node while satisfying these resource constraints.

F-TSPTW relaxation of (FP) can be represented as an ESPPRC. Let us consider a machine k and the set of operations $\mathcal{O}_k$. A node is introduced for each operation of $\mathcal{O}_k$. An arc (u, v) is introduced between each pair of operations $(u = O_{ij}, v = O_{xy})$, with a cost $c_{uv} = -M$, where M is an arbitrary large constant. A single resource ($Q = 1$) is defined, with $l_{uv}^1 = s_{ixk} + p_{xy}$. The resource window for node $u = O_{ij}$ is set as $[a_u^1, b_u^1] = [r_{ij} + p_{ij}, d_{ij}]$. Two additional nodes are introduced for the origin and the destination. An arc with a resource consumption s_{0ik} is added between the origin and every operation node $u = O_{ij}$, while an arc with a resource consumption 0 is added between every operation node and the destination. Note that the triangular inequality is respected since we have $s_{ijk} \leq s_{izk} + s_{zjk} \implies s_{ijk} + p_j \leq s_{izk} + p_z + s_{zjk} + p_j$, for all distinct $i \in [0, n], j, z \in [1, n]$. Costs are defined such that the optimal ESPPRC solution visits all nodes if possible. Solving the ESPPRC thus permits to solve the F-TSPTW, as expected.

We solve the problem by using the dynamic algorithm proposed by Feillet et al [22] This algorithm follows the classical Bellman's algorithm. The principle is to associate a label with each possible partial path and to extend these labels checking the resource constraint, until the best feasible paths are obtained. Dummy resources (indicating the reachability of nodes) are introduced to preserve path elementarity. Dominance rules are used to compare labels and remove some of them. The algorithm is adapted to avoid extending a label for which a non-visited node is unreachable.

To speed up the resolution process, we use a lower bound computation for each generated label. Let t denote the time associated to a label corresponding to a partial path from the origin node to a node $v = O_{ij}$, i.e. $t = W_v^q$ for this label. Let $\overline{\mathcal{O}}_k \subset \mathcal{O}_k$ denote the set of operations on machine k still unvisited by the partial path. We can use consistency rule (30) adapted to set $\Omega \cup \{O_{ij}\}$. Then, it is not useful to extend the current label whenever $\exists \Omega \subseteq \overline{\mathcal{O}}_k$, $t + p_{ij} + p_\Omega + s_{f_{ij} \to \Omega \cup \{f_{ij}\}} > d_\Omega$. Since the number of generated labels can be very large we keep the complexity of the lower bound linear by considering only $\Omega = \overline{\mathcal{O}}_k$.

10 Branching scheme and dominance rule

Several branching schemes have been proposed for the standard job-shop problem [25]. For the SDST-JSP, Brucker and Thiele [13] consider blocks of operations on the critical path of a feasible schedule issued from the current node. Branching is based on the property that to improve the current feasible solution, at least one operation has to be scheduled either before all the other ones in its block (if it it not already the first one) or after all the other ones (if it it not already the last one in its block). Hence from a current selection, the child nodes are generated by numerating these different possibilities and by making the induced selection updates. Focacci et al [23] use a sequence-based branching scheme by memorizing which operation has been scheduled last on each machine. The branching scheme consist in selecting an unscheduled operation and perfom a binary branching for this operation. For the left branch, the operation is scheduled next in the sequence of its machine. For the right branch the operation is scheduled as a successor but not next of the last sequenced operation.

In this paper we use a mixed chronological/active branching scheme based on the current partial selection. Let $E_{ij} \subseteq E$ denote the set of undirected disjunctive edges connected to O_{ij} at the current node $\nu = (\mathcal{R}, \mathcal{D}, \mathcal{E})$. When the selection $\mathcal{E}$ is complete, we have $E_{ij} = \emptyset$, for each operation $O_{ij} \in \mathcal{O}$.

Let $O_{i^*j^*}$ denote the operation such that $E_{i^*j^*} \neq \emptyset$ and

$$r_{i^*j^*} = \min_{O_{ij} \in \mathcal{O}, E_{ij} \neq \emptyset} r_{ij} \tag{31}$$

Let $\mathcal{C}$ denote the set of operations "conflicting" with $O_{i^*j^*}$, i.e. such that:

$$\mathcal{C} = \{O_{ij} \in \mathcal{O} | E_{ij} \neq \emptyset, m_{ij} = m_{i^*j^*}, r_{ij} < r_{i^*j^*} + p_{i^*j^*} + s_{i^*im_{ij}}\} \tag{32}$$

The branching scheme we propose aims generates $|\mathcal{C}|$ nodes $\{\nu_{ij}\}_{O_{ij} \in \mathcal{C}}$ from the current node ν where $\nu_{ij} = (\mathcal{R}, \mathcal{D}, \mathcal{E} \cup \{(O_{ij}, O_{xy})\}_{O_{xy} \in \mathcal{C} \setminus O_{ij}})$.

It has to be underlined that such a conflict set definition may result in non active schedules since the conflict set is defined only with necessary conditions. Indeed, there may be some operations in $\mathcal{C}$ whose earliest start time is not fixed and may increase in the descendence of the current node. This cannot be the case for operation $O_{i^*j^*}$ because it has the smallest earliest start time among operations linked to undirected disjunctive edges. However considering an operation $O_{ij} \neq O_{i^*j^*}$ of set $\mathcal{C}$ then a child node of ν_{ij} for which r_{ij} becomes greater than or equal to $r_{i^*j^*} + p_{i^*j^*} + s_{i^*im_{ij}}$

can be erased. This can be simply achieved by setting $d_{ij} = r_{i^*j^*} + p_{i^*j^*} + s_{i^*im_{ij}} + p_{ij} - 1$.

11 Computational Results

In this Section, we present computational experiments conducted to evaluate the quality of our approach. For this purpose, we use the benchmark instances from Brucker and Thiele [13]. These instances are issued from the classical Lawrence instances [28] devoted to the Job Shop Problem, introducing setup times. Each instance is characterized by a number of machines, a number of jobs to be scheduled and a number of setup types for the operations. These three parameters define a triplet with the format (machines $\times$ jobs $\times$ types). There are 15 instances with sequence dependent setup times (named t2-ps01 to t2-ps15). Instances t2-ps01 to t2-ps05 are of type $5\times10\times5$ (small instances). Instances t2-ps06 to t2-ps10 are of type $5\times15\times5$ (medium instances). Instances t2-ps11 to t2-ps15 are of type $5 \times 20 \times 10$ instances (large instances). Brucker and Thiele [13] remarked that the corresponding Lawrence instances without setup times (LA01 to LA15) were all easily solved by their branch and bound method. On the opposite, only the small sequence-dependent setup time instances (t2-ps01 to t2-ps05) were solved to optimality.

Since the branch and bound method is designed to solve problem $FP(T)$ we apply it different times to find the optimal solution of (P) according to principle (6). Due to the succesive runs, we set a limit NBB to the number of nodes of the branch and bound described in Section 6 and we also apply the following refinements:

(i) To keep the time spent at each branch and bound node reasonable, we set a limit NDP on the number of iterations (label extensions) of the dynamic programming algorithm used to solve the TSPTW relaxations (see Section 9). When this limit is reached, nothing can be deduced from SolveTSPTW and branching is necessary.

(ii) For diversification purposes, we also use a randomized version of the heuristic SerialSGS described in Section 7. This randomized version selects at step 4 an operation different than the one with the minimal priority according to a random factor. At each node, SerialSGS is called 6 times with the priority values given in Section 7 but for both the deterministic and randomized versions.

(iii) The heuristic keeps track of the best found solution, even if its makespan is greater than the current threshold T. Furthermore, each time the heuristic improves the best solution found so far, the randomized version of SerialSGS is called NH times with the current priority value, for intensification purposes.

Then the following strategies have been implemented to solve (P):
strategy 1: A binary search is performed between an initial trivial lower bound (set to the longest job duration) and an initial upper bound set of one call of the serial SGS with the earliest start priority rule. Parameters are $NBB = 500000$, $NDP = 1500$ and $NH = 50000$.
strategy 2: An increasing linear search is performed, starting from the same initial trivial lower bound as for strategy 1 with parameters $NBB = 20 \times 10^6$, $NDP = 3000$, $NH = 50000$.

The algorithms are coded in C++ and the tests are carried out on a PC with AMD64 architecture under Linux. In Table 1, we give the results of strategy 1. For each instance, we give the best found lower (LB1) and upper (UB1) bounds, the total number of branch and bound nodes (#nodes), the total CPU time in seconds (CPU), the initial lower bound value (LB0), the initial upperbound value (UB0), the number of calls of BranchAndBound by the binary search (IT) and the number of times a TSPTW relaxation could not be solved because the iteration limit NDP has been reached (TO).

As seen in Table 1, Strategy 1 solves to optimality 8 problems out of 15: all the 5 small instances, 3 out of 5 medium instances and none of the large instances. The previously unsolved problem t2-ps06 is closed with an optimal value of 1009, in 6157 seconds. The large CPU time values for the medium and large instances are due to the very large initial gap (since we did not take the best known lower and upper bound as initial bounds, but rather trivial ones) and to the maximal number of nodes which is reached for all values between the best found lower and upper bounds. All the best known lower bounds (see also Table 4) of the previously unsolved instances are improved. Two best known upper bounds are improved for instances t2-ps06 and t2-ps14 (see also Table 5).

In Table 2, we give the results of strategy 2 giving the same information as for strategy 1, but omitting the number of iterations and the number of time outs for the dynamic programming method. Note the number of iterations is obtained here by substracting the initial lower bound from the obtained lower bound. For the computation of lower bounds, strategy 2 obtains better results than strategy 1 (see also Table 4). Indeed, strategy

2 solves all the small instances and 4 out of 5 medium instances. The previously unsolved problem t2-ps08 is closed with an optimal value of 963 in 349923 seconds. None of the large instances can be solved to optimality but all lower bounds are significantly improved, at the expense of very large computational times. Except for instance t2-ps08, the upper bounds found by strategy 2 are not as good as the ones obtained by strategy 1 (see also Table 5). This is not surprising since the linear search is performed increasingly from the lower bound. Consequently, the heuristics are never guided by a feasible makespan value, except for the optimal one.

Besides the results shown in Tables 1 and 2, we notice that solving the TSPTW problems at each node appears crucial for the efficiency of the method. To take a single example, removing the TSPTW resolution (i.e. only the constraint propagation algorithms are applied) for the t2-ps05 instance increases the number of nodes up to 389992 nodes and the CPU time to 346.71 whereas the problem is solved in 3345 nodes and 16.2 seconds if the TSPTW relaxation is used. Martin and Shmoys [29] notice that for the standard job-shop problem, solving exactly the one-machine problem at each node with the Carlier's algorithms brings only a little improvement compared to using edge finding and immediate selections. This seems to be no more the case when sequence-dependent setup times are introduced.

Table 3 provides the results of the stragegy 1-based exact method on the corresponding job-shop instances without setup times [28]. All of these instances are easily solved by our method, which confirms the difficulty of the setup times constraints, as stated by Brucker and Thiele [13].

Table 4 gives the result of several lower bounds on the 15 SDST-JSP instances, to evaluate the different components of our method and to make comparisons with other lower bounds. Column LB0 recalls the value of the trivial lower bound based on longest path computations in the precedence graph. Column Shaving gives the lower bound obtained if only the shaving process is applied, i.e. the greatest value for which shaving can prove infeasibility. The lower bound value and the CPU times are given. Column TSPTW gives the lower bound obtained by applying only shaving followed by the resolution of the TSPTW relaxation. This is done by setting the branch and bound node limit to 1 and by setting no limit to the number of label extensions of the dynamic programming algorithm. The lower bound value and the maximal experienced number of iterations of the dynamic programming algorithm are provided. Column LB1 and LB2 recall the results of the best lower bounds by the two strategies we have implemented. We provide the results obtained by the root node lower bound of the branch and bound method proposed by Brucker and Thiele [13] (BT96), the root

node and the best lower bound of the branch and bound method proposed in Artigues et al [3] (ABF04). The root node lower bound of Artigues et al [3] corresponds to the resolution of the TSPTW relaxation without tightening the time window by the shaving process. The best lower bound of [3] was obtained by branch and bound.

The following ranking can be expected and is verified:
LB(BT96) < LB root(ABF04), LB(SHAVING) < LB(TSPTW) < LB1 < LB2. Note, however, that since there is a number of dynamic programming iterations limited to 1500 in our implementation of strategy 1, the TSPTW bound is not obtained without branching in our methods. One can observe that truncating the dynamic programming was necessary, due to an important number of iterations (label extensions) for the medium instances. The TSPTW bound could not even be obtained in a reasonable amount of time for the large instances. The shaving based-bound is tight 2 times and is generally better than the TSPTW bound without shaving (LB root ABF04) and than the Brucker and Thiele [13] bound (LB root BT96), which indicates the good results of the constraint propagation algorithms and confirms the power of the shaving technique already experienced for the standard job-shop problem [37, 29]. The computational time of the shaving algorithm is rather high but one could considerably accelerate it by performing a bisection search on the time window as described by Martin and Shmoys [29]. As a main result of our approach, strategy 2 obtains all the best known lower bounds on all instances, outperforming all other methods.

Table 5 gives the results of the state-of-the-art method in terms of the best obtained feasible solutions. BT96, FNL00, ALA05, ABF04, ABF05, BSV05 stand for the solutions found by [13, 23, 5, 3, 4, 8], respectively. The results show that the heuristic solutions found by our algorithm are not so far from the best known solutions mostly obtained by the shifting bottleneck heuristic of Balas et al [8] with reasonable computational requirements. We improve upon the results of Balas et al [8] for small instance t2-ps05, for 3 medium instances out of 5 (t2-ps06, t2-ps07 and t2-ps08) and for 1 large instance out of 5 (t2-ps14). The results of Balas et al [8] are better than ours for instance t2-ps09 and for all large instances except t2-ps14. As already underlined, the solutions found by strategy 1 on large instances are better than the one obtanied by strategy 2. Note the initial heuristic (UB0), which carries out a single run of the serial SGS with the earliest-start priority-rule, obtains better results than the truncated branch-and-bound methods of Brucker and Thiele [13] and Focacci et al [23] on 1 medium instance and on 3 large instances. It follows, as already mentionned by Artigues et al [5], that the latter methods could be highly strengthened by integrating the

serial SGS in the set of heuristics they use.

Last, Table 6 gives the evolution the gap between the best known lower and upper bound accross all concerned studies since the paper of Brucker and Thiele [13] as well as the number of improved lower bounds, upper bounds and closed instances obtained by each previous work. The results show that the present study has significantly reduced the gap on all unsolved instances. The gap is reduced to less than 1% on the unsolved medium instance and to less that 8% on all the large instances. Solving all the medium sized instances will be probably achieved in a near future. However the large instances remain challenging.

12 Concluding remarks

We have proposed a new exact method to solve the job-shop problem with sequence-dependent setup times which significantly reduces all gaps for the unsolved problem instances proposed by Brucker and Thiele [13]. The results are obtained thanks to a cooperation between constraint propagation techniques extended to setup times and a truncated dynamic programming-based resolution of TSPTW relaxations. The feasible solutions are obtained by a serial schedule generation scheme with a priority-rule based on the solutions of the TSPTW relaxations.

The cooperation between constraint progagation and TSPTW relaxations could be further increased, for instance by solving smaller TSPTW corresponding to special subsets of operations sequenced on the same machine or conversely by designing time windows update methods based on the analysis of the TSPTW. The feasible solutions of the TSPTW could also be maintained by local search algorithms adapting to the branching decisions. In the near future, we will focus on improvements that can be brought to speed up the resolution process of the dynamic programmming algorithm. A better resolution of the TSPTW could be the key to solve larger instances.

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for jobshop scheduling. *Management Science*, 34:391–401, 1988.

[2] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27:219–239, 1999.

[3] C. Artigues, S. Belmokhtar, and D. Feillet. A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In J.C.-Régin and M. Rueher, editors, *1st International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Note in Computer Science 3011, pages 37–49. Springer, 2004.

[4] C. Artigues, F. Buscaylet, and D. Feillet. Lower and upper bound for the job shop scheduling problem with sequence-dependent setup times. In *proceedings of the second Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'2005)*, New York, 2005.

[5] 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.

[6] E. Balas. New classes of efficiently solvable generalized traveling salesman problems. Technical Report #MSRR-615, Graduate School of Industrial Administration, Carnegie Mellon University, 1996.

[7] E. Balas and N. Simonetti. Linear time dynamic programming algorithms for new classes of restricted TSPs. *INFORMS Journal on Computing*, 13:56–75, 2001.

[8] E. Balas, N. Simonetti, and A. Vazacopoulos. Job shop scheduling with setup-times, deadlines and precedence constraints. In *proceedings of the second Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'2005)*, New York, 2005.

[9] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling*, volume 39 of *International Series in Operations Research & Management Science*. Springer, 2001.

[10] J. Blazewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33, 1996.

[11] P. Brucker, P. Jurisch, and A. Krämer. The job-shop problem and immediate selection. *Annals of Operations Research*, 50:73–114, 1994.

[12] P. Brucker, P. Jurisch, and B. Sievers. A fast branch and bound algorithm for for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[13] P. Brucker and O. Thiele. A branch and bound method for the general-shop problem with sequence-dependent setup times. *Operations Research Spektrum*, 18:145–161, 1996.

[14] F. Buscaylet and C. Artigues. A fast tabu search method for the job-shop problem with sequence-dependent setup times. In *Metaheuristic International Conference MIC'2003*, 2003.

[15] M. A. B. Candido, S.K. Khator, and R. M. Barcias. A genetic algorithm based procedure for more realistic job shop scheduling problems. *International Journal of Production Research*, 36(12):3437–3457, 1998.

[16] J. Carlier. The one machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[17] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

[18] I.-C. Choi and O. Korkmaz. Job shop scheduling with separable sequence-dependent setups. *Annals of Operations Research*, 70:155–170, 1997.

[19] I.-N. Choi and D.-S. Choi. A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers and Industrial Engineering*, 42:43–58, 2002.

[20] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Company, 1967.

[21] S. Demassey, C. Artigues, and P. Michelon. Constraint propagation-based cutting planes : an application to the resource-constrained project scheduling problem. *INFORMS Journal on computing*, 17(1):52–65, 2005.

[22] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[23] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 92–101, 2000.

[24] F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the tsptw. *INFORMS Journal on Computing*, 14:403–417, 2002.

[25] A. S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390–434, 1999.

[26] S.C. Kim and P.M. Bobrowski. Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research*, 32(7):1503–1520, 1994.

[27] R. Kolisch. serial and parallel resource-constrained project scheduling methods revisited: theory and computation. *European Journal of Operational Research*, 90:320–333, 1996.

[28] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1984.

[29] P. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization IPCO'96*, pages 389–403, Vancouver, British Columbia, Canada, 1996.

[30] S. J. Mason, J. W. Fowler, and W. Matthew Carlyle. A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, 5(3):247 – 262, 2002.

[31] J. A. Noivo and H. Ramalhinho-Lourenço. Solving two production scheduling problems with sequence-dependent set-up times. Technical

Report 138, Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, 1998.

[32] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.

[33] W. P. M. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, 1994.

[34] I.M. Ovacik and R. Uzsoy. A shifting bottleneck algorithm for scheduling semiconductor testing operations. *Journal of Electronics Manufacturing*, 2:119–134, 1992.

[35] I.M. Ovacik and R. Uzsoy. Exploiting shop floor status information to schedule complex job shop. *Journal of manufacturing systems*, 13:73–84, 1994.

[36] I.M. Ovacik and R. Uzsoy. Rolling horizon algorithms for a single machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 32(6):1243–1263, 1994.

[37] L. Peridy. *Le problème de job-shop : arbitrages et ajustements*. PhD thesis, Université de Technologie de Compiègne, 1996.

[38] B. Roy and B. Sussman. Les problèmes d'ordonnancement avec contraintes disjonctives. Technical Report Note DS no 9bis, SEMA, Paris, 1964.

[39] M.W.P. Savelsberg. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[40] J.M.J. Schutten. Practical job shop scheduling. Technical Report LPOM-95-12, Laboratory of Production and Operations Management, Department of Mechanical Engineering, University of Twente, The Netherlands, 1995.

[41] X. Sun and J.S. Noble. A modified shifting bottleneck approach to job shop scheduling with sequence dependent setups. *Journal of Manufacturing Systems*, 18(6):416–430, 1999.

[42] P. Torres and P. Lopez. Overview and possible extensions of shaving techniques for job-shop problems. In *Proceedings of the Workshop*

on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR'00, pages 181–186, Paderborn, Germany, 2000.

[43] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8:302–317, 1996.

[44] P. Vilím and R. Barták. Filtering algorithms for batch processing with sequence dependent setup times. In Ghallab, Hertzberg, and Traverso, editors, *Proceedings of The Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, pages 312–320. AAAI Press, 2002.

[45] J.K. Wilbrecht and W.B. Prescott. The influence of setup time on job shop performance. *Management Science*, 16(4):B274–B280, 1969.

[46] C. Zhou and P.G. Egbelu. Scheduling in manufacturing shop with sequence-dependent setups. *Robotics and Computer Integrated Manufacturing*, 5:73–81, 1989.

Table 1: Best lower and upper bounds obtained by strategy 1

Problem	LB1	UB1	#nodes	CPU	LB0	UB0	#IT	#TO
t2-ps01	**798***	**798***	173697	400.6	433	844	8	747
t2-ps02	**784***	**784***	24380	105.1	434	992	7	2810
t2-ps03	**749***	**749***	150885	352.6	359	946	9	8323
t2-ps04	**730***	**730***	2414	16.3	399	921	7	215
t2-ps05	**691***	**691***	3345	16.2	390	733	5	182
t2-ps06	**1009***	**1009***	1143776	6156.2	433	1120	8	150768
t2-ps07	**970***	**970***	1533045	10012.6	416	1129	10	336768
t2-ps08	946	982	2583663	22716.5	399	1066	10	507641
t2-ps09	1049	1061	2210868	26754.5	412	1174	9	490508
t2-ps10	**1018***	1047	1499685	6391.1	463	1187	9	20895
t2-ps11	1373	1494	4512218	39489.9	483	1719	13	671506
t2-ps12	1219	1381	3072119	26678.8	498	1425	10	411984
t2-ps13	1317	1457	5414824	50336.8	462	1531	14	864255
t2-ps14	1429	**1483**	3583123	37236.1	463	1549	10	509888
t2-ps15	1392	1661	4342505	46590	431	1749	13	696812

Value in bold: best known result on the instance is reached.
Value underlined: previously best known result on the instance is improved.
*tight bound

Table 2: Best lower and upper bounds obtained by strategy 2

Problem	LB2	UB2	#nodes	CPU	LB0	UB0
t2-ps01	**798***	**798***	1	56.7	433	844
t2-ps02	**784***	**784***	9498	242.3	434	992
t2-ps03	**749***	**749***	181090	699.3	359	946
t2-ps04	**730***	**730***	3594	251.6	399	921
t2-ps05	**691***	**691***	2918	58.2	390	733
t2-ps06	**1009***	**1009***	256520	1797.6	433	1120
t2-ps07	**970***	**970***	71106	781.8	416	1129
t2-ps08	**963***	**963***	33491952	349923	399	1066
t2-ps09	**1051**	1061	20105686	169582	412	1174
t2-ps10	**1018***	**1018***	227	35.1	463	1187
t2-ps11	**1395**	1617	57981046	916833	483	1719
t2-ps12	**1242**	1424	69387629	914086	498	1425
t2-ps13	**1342**	1457	58651163	895059	462	1531
t2-ps14	**1432**	1499	19999854	306899	463	1549
t2-ps15	**1406**	1671	52337255	792196	431	1749

Value in bold: best known result on the instance is reached.
Value underlined: previously best known result on the instance is improved.
*tight bound

Table 3: Results of strategy 1 on the instances with no setuptimes

Problem	Opt	#nodes	CPU	#IT	#TO
LA01 (t2-ps01)	**666***	1	7.5	7	1
LA02 (t2-ps02)	**655***	22150	70.1	8	752
LA03 (t2-ps03)	**597***	2	8.5	8	1
LA04 (t2-ps04)	**590***	42	13	8	8
LA05 (t2-ps05)	**593***	1	2.2	8	1
LA06 (t2-ps06)	**926***	1	4.6	6	1
LA07 (t2-ps07)	**890***	107	17.2	7	74
LA08 (t2-ps08)	**863***	1	27.42	8	1
LA09 (t2-ps09)	**951***	1	39.4	9	1
LA10 (t2-ps10)	**958***	1	32.2	8	1
LA11 (t2-ps11)	**1222***	1	14.6	9	1
LA12 (t2-ps12)	**1039***	15	112.6	9	15
LA13 (t2-ps13)	**1150***	19	29.33	8	19
LA14 (t2-ps14)	**1292***	1	207.1	8	1
LA15 (t2-ps15)	**1207***	1080	92.3	10	366

Table 4: Lower bound comparison

Problem	LB0	Shaving		TSPTW		LB1	LB2	LB root	LB root	LB
		LB	(CPU)	LB	(DPIT)			BT96	ABF04	ABF0
t2-ps01	433	781	(9.5)	796	(10847)	**798***	**798***	756	796	**798**
t2-ps02	434	745	(7.3)	745	(13667)	**784***	**784***	705	715	**784**
t2-ps03	359	710	(6.7)	710	(9079)	**749***	**749***	658	678	**749**
t2-ps04	399	707	(9.7)	707	(10945)	**730***	**730***	627	647	**730**
t2-ps05	390	687	(4.2)	690	(9049)	**691***	**691***	653	671	**691**
t2-ps06	433	1006	(52.2)	1006	(239221)	**1009***	**1009***	986	996	996
t2-ps07	416	**970***	(20.0	**970***	(927515)	**970***	**970***	940	927	**970**
t2-ps08	399	923	(14.7)	930	(1130061)	946	**963***	913	923	923
t2-ps09	412	1011	(16.1)	1012	(858939)	1049	**1051**	1001	1012	1037
t2-ps10	463	**1018***	(19.3)	**1018***	(788711)	**1018***	**1018***	1008	**1018***	**1018**
t2-ps11	483	1360	(83.5)	—	—	1373	**1395**	1322	NA	NA
t2-ps12	498	1175	(145.8)	—	—	1219	**1242**	1139	1159	1159
t2-ps13	462	1280	(68.6)	—	—	1317	**1342**	1250	1250	1250
t2-ps14	463	1412	(181.8)	—	—	1429	**1432**	1402	NA	NA
t2-ps15	431	1357	(182.4)	—	—	1392	**1406**	1307	NA	NA

Value in bold: best known result on the instance is reached.
Value underlined: previously best known result on the instance is improved.
*tight bound

Table 5: Upper bound comparison

Problem	UB0	BT96	FNL00	ALA05	ABF04	ABF05	BSV05	UB1	UB2
t2-ps01	844	**798***	**798***	818	**798***	**798***	**798***	**798***	**798***
t2-ps02	992	**784***	**784***	829	**784***	**784***	**784***	**784***	**784***
t2-ps03	946	**749***	**749***	782	**749***	771	**749***	**749***	**749***
t2-ps04	921	**730***	**730***	745	**730***	743	**730***	**730***	**730***
t2-ps05	733	**691***	**691***	704	**691***	693	693	**691***	**691***
t2-ps06	1120	1056	NA	1026	1026	1026	1018	**1009***	**1009***
t2-ps07	1129	1087	NA	1033	**970***	1022	1003	**970***	**970***
t2-ps08	1066	1096	NA	1002	1002	994	975	982	**963***
t2-ps09	1174	1119	NA	**1060**	**1060**	**1060**	**1060**	1061	1061
t2-ps10	1187	1058	NA	1036	**1018***	**1018***	**1018***	1047	**1018***
t2-ps11	1719	1658	NA	1478	NA	1509	**1470**	1494	1617
t2-ps12	1425	1528	1448	1319	1319	1305	**1305**	1381	1424
t2-ps13	1531	1549	1658	1439	1439	1439	**1439**	1457	1457
t2-ps14	1549	1592	NA	1492	NA	1492	1485	**1483**	1499
t2-ps15	1749	1744	NA	1559	NA	1556	**1527**	1661	1671

Value in bold: best known result on the instance is reached.
Value underlined: previously best known result on the instance is improved.
*tight bound

Table 6: Chronological Evolution of the gap $(1 - LB/UB) \times 100$ for the SDST-JSP instances

Problem	BT96	FNL00	ALA05	ABF04	ABF05	BSV05	Our results
t2-ps01	0						
t2-ps02	0						
t2-ps03	0						
t2-ps04	0						
t2-ps05	0						
t2-ps07	13.52	—	9.00	0			
t2-ps10	4.73	—	2.70	0			
t2-ps06	6.62	—	3.90	2.92	—	2.16	0
t2-ps08	16.70	—	8.88	7.88	7.14	5.33	0
t2-ps09	10.54	—	5.56	2.17	—	—	0.85
t2-ps11	20.27	—	10.55	—	—	10.06	5.10
t2-ps12	25.46	21.33	13.64	12.13	11.18	—	4.83
t2-ps13	19.30	—	13.13	—	—	—	6.74
t2-ps14	11.93	—	6.03	—	—	5.59	3.44
t2-ps15	25.05	—	16.16	—	16.00	14.40	7.92
LB impr	15	0	0	6	0	0	8
UB impr	15	1	10	2	3	5	3
closed	5	0	0	2	0	0	2