A New Exact Solution Algorithm for the Job Shop Problem with Sequence-Dependent Setup Times

Christian Artigues[1], Sana Belmokhtar[2], and Dominique Feillet[1]

[1] Laboratoire d'Informatique d'Avignon, FRE CNRS 2487, Université d'Avignon, Agroparc BP 1228, 84911 Avignon Cedex 9, France
[2] MSGI-G2I, Ecole des Mines de Saint Etienne 158, cours Fauriel, 42023 Saint Etienne Cedex 2, France

Abstract. We propose a new solution approach for the Job Shop Problem with Sequence Dependent Setup Times (SDST-JSP). The problem consists in scheduling jobs, each job being a set of elementary operations to be processed on different machines. The objective pursued is to minimize the completion time of the set of operations. We investigate a relaxation of the problem related to the traveling salesman problem with time windows (TSPTW). Our approach is based on a Branch and Bound procedure, including constraint propagation and using this relaxation. It compares favorably over the best available approaches from the literature on a set of benchmark instances.

1 Introduction

In this work, we consider the Job Shop Problem with Sequence Dependent Setup Times (SDST-JSP). The Job Shop Problem is used for modeling problems where a set of jobs, consisting of a sequence of elementary operations to be executed on distinct machines, has to be scheduled. This problem is widely investigated in the literature and many efficient approaches exist for its resolution (see, e.g., Blazewicz *et al.* [5], Nowicki and Smutnicki [13] or Vaessens *et al.* [15]). The SDST-JSP is a variant problem where machines have to be reconfigured between two successive operations. The most common objective is to minimize the completion time of the set of operations, i.e., the so-called makespan.

The SDST-JSP is trivially NP-hard in the strong sense since it admits the Job Shop Problem as a special case, which is also NP-hard in the strong sense. However, despite its similarities with the Job Shop Problem, few works have been devoted to its solution. Some heuristic solution approaches have been proposed in the literature (Choi and Choi [8], Artigues and Buscaylet[3], Artigues *et al* [2]), but no exact solution algorithm strictly addresses this problem. Actually, Brucker and Thiele [6] describe a Branch and Bound algorithm for the General-Shop Problem (GSP), which admits the SDST-JSP as a special case. Also, Focacci

et al. [9] propose a Branch and Bound scheme for a variant problem where the machines used for operations are not fixed but to be chosen in a given subset.

In this paper, we also propose a Branch and Bound procedure using constraint propagation techniques. A relaxation is introduced, based on the search of feasible solutions for Traveling Salesman Problem with Time Windows (TSPTW) instances. Note that our approach presents many similarities with Focacci *et al.* [9]'s one, except that Focacci *et al.* [9] prefer to relax subtour constraints and tackle assignment problems. Indeed assignment problems can be solved in polynomial time, while the TSPTW is NP-hard in the strong sense. The assignment problem relaxation is embedded into a global constraint associated with a reduced cost-based filtering algorithm. In our algorithm the TSPTW is considered without relaxation but a long term memory is introduced for limiting computing times. It memorizes TSPTW instances solved throughout the search tree and permit the quick obtaining of feasible solutions in many cases. The relaxation is used to compute an initial lower bound and to test feasibility at the nodes of the search in a dichotomy framework, but it does not provide additional domain filtering.

The problem and the model used are described in section 2. Section 3 then presents the backbone of the algorithm, i.e., the Branch and Bound scheme. The relaxation is described in section 4. Numerical experiments conclude the paper in section 5.

2 Presentation and mathematical formulation for the SDST-JSP

We consider a set $J = \{J_1, \ldots, J_n\}$ of n jobs. Jobs have to be processed on a set $M = \{M_1, \ldots, M_m\}$ of m resources (machines). Each job is made up of m operations $O_{i1}, \ldots, O_{im}$, to be processed in this order, without overlapping or preemption. The set of all operations is noted O and its size is $N = n \times m$. Operation $O_{ij} \in O$ requires machine $m_{ij} \in M$ and necessitates processing time $p_{ij} \geq 0$. Machines admit at most one operation at a time.

A set of setup types σ is defined, with a matrix $(|\sigma| + 1) \times (|\sigma|)$ of setup times noted s. To each operation O_{ij} is associated setup type $\sigma_{ij} \in \sigma$. A setup time $s_{\sigma_{ij}\sigma_{kl}}$ is then necessary between two consecutive operations O_{ij} and O_{kl} on the same machine. Also, an initial setup times $s_{0\sigma_{ij}}$ is necessary if O_{ij} is the first operation on machine m_{ij}. Finally, we assume that setup times satisfy the triangle inequality.

The objective pursued in this work is the minimization of the makespan, that is the completion time of the last operation processed, even if other objectives could have been considered.

A solution of the SDST-JSP can be represented with the help of the so-called job and machine Gantt's diagrams. An example of such diagrams is given in figure 1 for some instance of the SDST-JSP with 3 jobs and 3 machines.

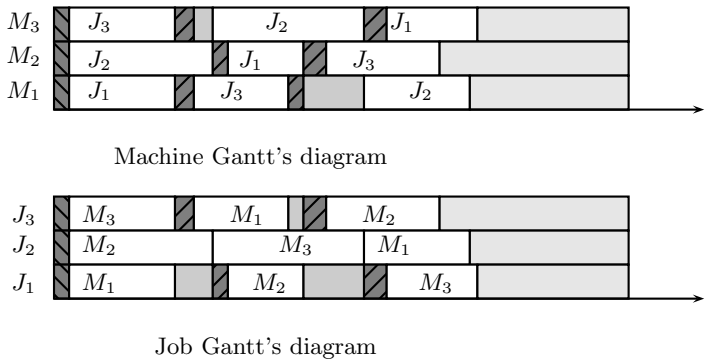

Fig. 1. Gantt's diagrams

On these diagrams, we represent setup times with hatched rectangles. Full rectangles stand for idle times. The two diagrams respectively represent a solution from both machine and job views.

Using the constraint programming framework, a simple model can easily be proposed for the SDST-JSP. For each operation O_{ij}, we introduce a variable S_{ij} indicating the starting time of the operation. A variable C_{max} is also introduced to represent the value of the makespan. The model is then:

$$\text{minimize } C_{max} \tag{1}$$

subject to

$$C_{max} \geq S_{im} + p_{im} \quad (i = 1, \ldots, n) \tag{2}$$

$$S_{ij} + p_{ij} \leq S_{i(j+1)} \quad (O_{ij} \in O; j < m) \tag{3}$$

$$(S_{ij} \geq S_{hl} + p_{hl} + s_{\sigma_{hl}\sigma_{ij}}) \text{ or}$$
$$(S_{hl} \geq S_{ij} + p_{ij} + s_{\sigma_{ij}\sigma_{hl}}) \quad (O_{ij}, O_{hl} \in O; m_{ij} = m_{hl}) \tag{4}$$

$$S_{ij} \geq S_{0\sigma_{ij}} \quad (O_{ij} \in O) \tag{5}$$

$$S_{ij} \in \mathbb{N} \quad (O_{ij} \in O) \tag{6}$$

$$C_{max} \in \mathbb{N} \tag{7}$$

Constraints (2) state C_{max} as the makespan in optimal solutions. Constraints (3) represent the precedence constraints between the successive operations of

the same job. Disjunctive constraints between operations processed on a same machine are enforced with constraints (4). Setup times between operations on a same machine appear in these constraints, while initial setup times require a last set of constraints (5).

A useful tool for the solution of scheduling problems is the so-called disjunctive graph. 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 coresponds to a unique semi-active schedule 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, the SDST-JSP reduces exactly to 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, that is scheduling where operations are started as soon as possible (when disjunctive and conjunctive constraints are satisfied), to find an optimal solution. An example of a disjunctive graph is presented in figure 2.

3 Branch and Bound scheme

The Branch and Bound technique is an approach based on a non-exhaustive enumeration of the solutions of a problem. Two main features of the technique are the branching scheme and the bounding scheme. Branching replaces a problem (represented by a node of the search tree) with several new problems of reduced sizes (included in the tree as descendant nodes). Bounding permits the pruning of nodes for which it appears that the associated problems do not contain any optimal solution.

Our algorithm is based on a Branch and Bound embedded in a dichotomy framework, the Branch and Bound being limited to the search of a feasible

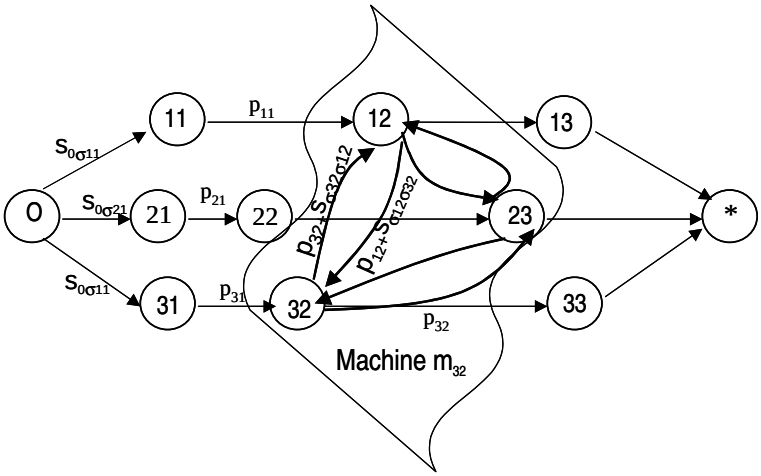

Fig. 2. Disjunctive graph for the SDST-JSP

solution. An initial lower bound LB is obtained by the relaxation of precedence constraints (3) - see section 4 for details. An initial upper bound UB is obtained using an existing heuristic solution approach. At each step of the dichotomy a value L is chosen in the interval $[LB, UB]$ that sets an upper limit for C_{max}. Depending whether a solution is found or not using the Branch and Bound, UB is fixed to the value of this solution or LB is fixed to L and the algorithm iterates.

In the following subsections, we focus on the Branch and Bound algorithm implemented in the dichotomy framework.

3.1 Branching scheme

Brucker and Thiele [6] base their branching scheme for the solution of the General Shop Problem on the disjunctive graph. Branching corresponds to choosing a direction for a disjunctive arc, until a complete selection is obtained.

We also base our branching on the disjunctive graph, but choosing a machine randomly and then the operations to be assigned on this machine. Each node of the search tree then provides a partial selection, where the direction of every disjunctive arcs issued from the chosen operation is fixed. The operations are taken in the increasing order of earliest start times and, in case of a tie, in the increasing order of latest finishing time. In our implementation, the selected machine is scheduled entirely before selecting the next one.

3.2 Constraint propagation

The main objective of constraint propagation is to determine disjunctive arcs for which a single direction is met in every feasible solutions (issued from the current node of the search tree). Indeed, propagation will permit to drop from decision variable domains values leading to unfeasible solutions. We perform propagation using the following filtering algorithms, that are detailed in Baptiste *et al.* [4].

Precedence constraints Each node of the search tree corresponds to a partial selection in the disjunctive graph $G = (V, U, E)$, where U is the set of conjunctive arcs and disjunctive arcs for which a direction has been chosen. Consistency is ensured at the bounds of the domains of variables S_{ij} (with $O_{ij} \in O$) using constraints $S_{i_2j_2} \geq S_{i_1j_1} + w_{i_1j_1,i_2j_2}$ $(O_{i_1j_1}, O_{i_2j_2}) \in U$, where $w_{i_1j_1,i_2j_2}$ is the weight of the arc in the graph. This propagation is performed by computing longest paths in (V, U).

Disjunctive constraints Using domains of variables S_{ij} (with $O_{ij} \in O$) and disjunctive constraints between operations on a same machine, it might be possible to deduce that an operation is necessarily processed before another one. Such a deduction corresponds to setting a direction of a disjunctive arc in the disjunctive graph.

For every operation $O_{ij} \in O$, we respectively note ES_{ij}, LS_{ij}, EF_{ij} and LF_{ij} the earliest starting time, latest starting time, earliest finishing time and latest finishing time of the operation, that are directly deduced from S_{ij}. Let O_{ij} and O_{hl} be two operations requiring the same machine. If $EF_{hl} + s_{\sigma_{hl}\sigma_{ij}} > LS_{ij}$, O_{hl} can not be processed before O_{ij}. This possibility is illustrated on figure 3, where $s_{\sigma_{hl}\sigma_{ij}} = 5$.

Propagation is activated as soon as the bounds of the domain of a variable S_{ij} are changed, for all the operations requiring the same machine m_{ij}.

Edge Finding This propagation scheme is a generalization of the preceding one. It permits to determine that an operation has to be processed before or after a set of other operations using the same machine. This propagation scheme is illustrated on figure 4, where operation O_{11} has to be processed after operations O_{22} and O_{31}, which cannot be deduced using the preceding propagation scheme.

It is worth mentioning that setup times penalize the effectiveness of this scheme. Indeed, while estimating the finishing time of a set of operations, the last operation of the set is not known and the setup time considered after the last operation of the set is the minimum setup time among all the operations of the set. Hence, some extensions of edge finding have been proposed [6, 16]. However, the classical version of edge finding ignoring setup times is considered in this study.

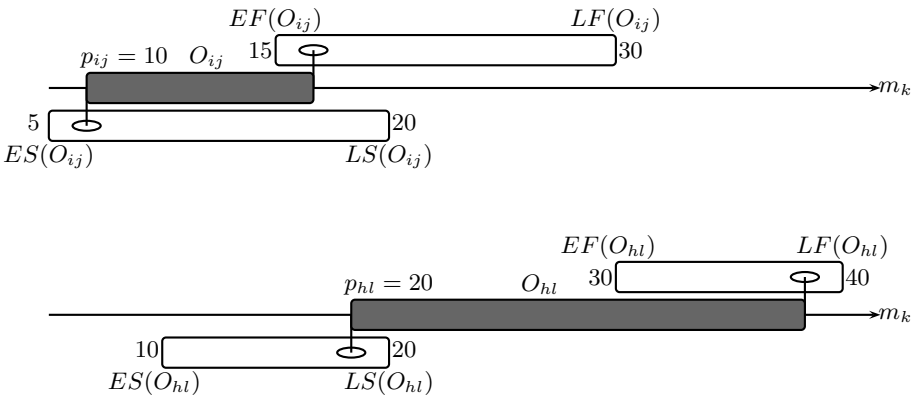

Fig. 3. Disjunctive constraints

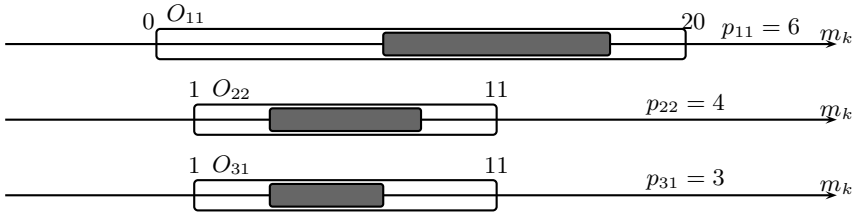

Fig. 4. Edge Finding

4 TSPTW relaxation

In this section, we describe the TSPTW relaxation used to compute an initial lower bound and, at the nodes of the dichotomizing search, for pruning as soon as unfeasibility is detected. This is performed through the relaxation of the precedence constraints in the model (1-7) of section 2. The new problem then decomposes into several TSPTWs.

The TSPTW consists in finding a route visiting one and only one time each vertex of a graph with a minimum cost, a travel cost being associated with the arcs of the graph. The visit of customers (vertices) is constrained to take place within given time windows. In the context of the SDST-JSP, customers stand for operations.

When relaxing precedence constraints in the model (1)-(7), m independent TSPTW instances appear, one for each machine. In the disjunctive graph, this relaxation is characterized by the elimination of the conjunctive arcs and by the appearance of m connected components. Hence, noting $TSPTW_k$ $(k = 1, \ldots, m)$ the TSPTW instance associated to machine k, finding k such that $TSPTW_k$ is unfeasible ensures that the current selection admits no feasible solution.

Figure 5 provides an example of this relaxation scheme with 3 jobs and 3 machines (where arcs issued from vertex 0 are omitted)

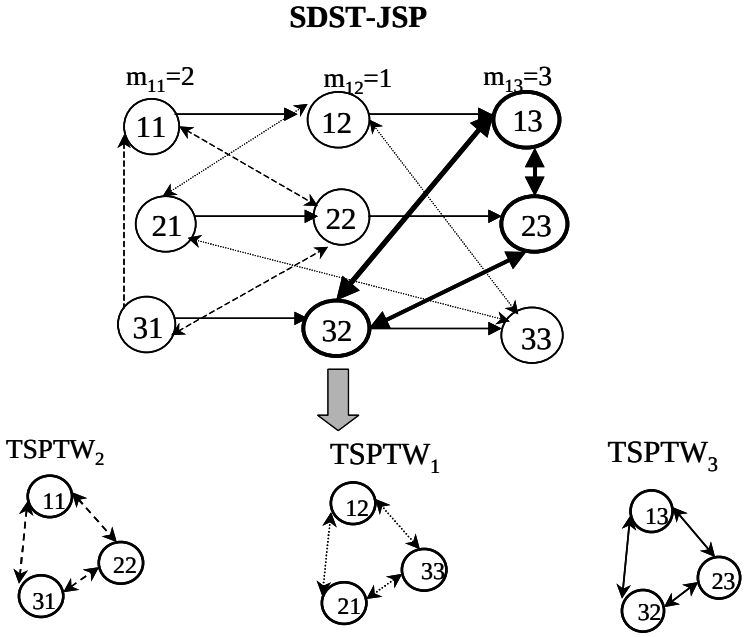

Fig. 5. Relaxation of the SDST-JSP into m TSPTWs

For every k in $\{1, \ldots, m\}$, we address the solution of $TSPTW_k$ with the following model. Let $O_k \subset O$ be the subset of operations to be considered in subproblem $TSPTW_k$ (operations on machine M_k). Variables S_{ij} of model (1)-(7)are used for operations $O_{ij} \in O_k$. Note that the domain of these variables is reduced through the branching decisions and the constraint propagation, which corresponds to time windows. We introduce a new variable $Tour_k$ for the objective function.

$$\text{minimize } Tour_k \tag{8}$$

subject to

$$Tour_k \geq S_{ij} + p_{ij} \quad (O_{ij} \in O_k) \tag{9}$$

$$S_{ij} \geq s_{0\sigma_{ij}} \quad (O_{ij} \in O_k) \tag{10}$$

$$S_{ij} \geq S_{hl} + p_{hl} + s_{\sigma_{hl}\sigma_{ij}}$$
$$\text{or } S_{hl} \geq S_{ij} + p_{ij} + s_{\sigma_{ij}\sigma_{hl}} \quad (O_{ij}, O_{hl} \in O_k; m_{ij} = m_{hl} = m_k) \tag{11}$$

$$S_{ij} \in \mathbb{N} \quad (O_{ij} \in O) \tag{12}$$

$$Tour_k \in \mathbb{N} \tag{13}$$

Before running the dichotomy, all $TSPTW_k$ are solved to optimality to compute a lower bound equal to the greatest optimal solution among the m problems. At each node of the dichotomizing search, a feasible solution for each $TSPTW_k$ is searched for using a commercial constraint programming package. In our implementation, each TSPTW is solved in turn as a one-machine scheduling problem with sequence-dependent setup times, release dates and deadlines. The disjunctive constraint and edge finding are set. A second dichotomizing binary search is used, as described in the chapter 7 of [12].

As soon as a k is found such that $TSPTW_k$ is unfeasible, the node is pruned. Otherwise, branching occurs and the search continues.

The main drawback using this relaxation is for computing time. It might be detrimental to solve m TSPTW at each node of the search tree, even if it provides a strong tool for pruning. In order to limit this drawback, we propose to memorize solutions of $TSPTW_k$ obtained so far during the search. Indeed, one might have to solve instances of the $TSPTW$ for which only some time windows have changed and for which sequences of customers previously found feasible might still be feasible.

Each time a solution is found, the ordered sequence of visited customers is memorized as a word in a dictionary. Then, when solving a $TSPTW_k$, the dictionary associated to machine M_k is first scanned to determine whether one of the sequences memorized is feasible for the current instance. If not, constraint programming is applied as explained above.

For each subproblem $TSPTW_k$, the dictionary is defined as a forest, as illustrated in figure 6. On this example, the dictionary contains sequences $J_3 \prec J_1 \prec J_2 \prec J_4$, $J_3 \prec J_1 \prec J_4 \prec J_2$ and $J_3 \prec J_2 \prec J_4 \prec J_1$. This structure is convenient as well for adding new words than for checking words for feasibility. For this last case especially, a simple depth first search can be implemented. When nodes are attained, earliest arrival times are computed, which permits to eventually backtrack. Hence, sequences beginning with the same unfeasible subsequence can be discarded simultaneously.

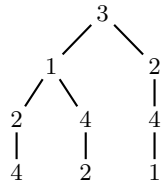

Fig. 6. Example of a dictionary for a $TSPTW_k$

5 Computational results

In this section, we present computational experiments conducted to evaluate the quality of our approach. For this purpose, we use benchmark instances from Brucker and Thiele [6]. These instances are issued from the classical Adams *et al.*'s [1] instances devoted to the Job Shop Problem, introducing setup. 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 $\times$ type. Computational experiments are realized on two sets of $5 \times 10 \times 5$ and $5 \times 15 \times 5$ instances. For comparison purpose, we use also two large $5 \times 20 \times 10$ instances.

Algorithms are implemented in C++ on a Pentium IV 2 GHz computer. ILOG Scheduler 5.2 [12] and ILOG Solver 5.2 [11] are used for constraint propagation and Branch and Bound.

Tables 1 and 2 compare the value of the bound we obtained at the root of the tree with the value obtained by Brucker and Thiele [6], respectively for the sets of $5 \times 10 \times 5$ and $5 \times 15 \times 5$ instances. Note that Brucker and Thiele [6] compute a lower bound by relaxing the GSP into a one machine problem and by using Jackson preemptive scheduling principle (JPS) and results from Carlier and Pinson [7]. Columns of these tables indicate successively the name of the instance, the value of Brucker and Thiele's bound, the value of our bound and an upper bound provided by Artigues *et al.* [2]. These two last values define the initial interval for dichotomy. In table 2, the computing time of the lower bound is given in seconds. These tables show that solving TSPTWs provide better bounds than the approach of Brucker and Thiele [6] for most of the instances. Although Focacci *et al* [9] do not explicitely provide a lower bound, the Assignment Problem solved in [9] is a relaxation of the TSPTW considered in the present paper and would consequently give a weaker (but faster) bound.

Table 3 compare the value of the makespan computed with our approach (ABF) with the value obtained by Brucker and Thiele [6] (BT) and Focacci *et al* [9] (FNL) for the smallest instances. A time limit of 7200 seconds is set for the computations for BT and ABF whereas a time limit of 1800 seconds is set for FNL. Columns indicate the name of the instance and, for each approach,

Table 1. Bound value at the root of the search tree for instances $5 \times 10 \times 5$

Instance	BT96	LB	UB
t2-ps01	756	796	818
t2-ps02	705	715	829
t2-ps03	658	678	782
t2-ps04	627	647	745
t2-ps05	653	671	704

Table 2. Bound value at the root of the search tree for instances $5 \times 15 \times 5$

Instance	BT96	LB	UB	CPU LB
t2-ps06	986	996	1026	1
t2-ps07	940	927	1033	0.9
t2-ps08	913	923	1002	3
t2-ps09	1001	1012	1060	19
t2-ps10	1008	1018	1036	141

the value obtained with the computing times. A last column ABF* indicates computing times when dictionaries are not used to memorize TSPTW solutions. Times are indicated in seconds. Times indicated for Brucker and Thiele [6]'s results are obtained on a Sun 4/20 station. Times indicated for Focacci *et al* [6]'s results are obtained on a Pentium II 200 MHz. Proven optimal solutions are indicated in bold. The table show that all $5 \times 10 \times 5$ instances are solved by our method in less than 500 seconds. The benefit of using the dictionary is clear. The comparison with BT and FNL approaches is not easy according to the considerable difference between the speeds of the computers. However, applying factors of 3 and 10 for CPU time comparisons with FNL and BT, respectively, our approach remains competitive.

Table 3. Makespan value for instances $5 \times 10 \times 5$

Instance	BT		FNL		ABF		ABF*
	Cmax	CPU (sec.)	Cmax	CPU (sec.)	Cmax	CPU (sec.)	
t2-ps01	**798**	502.1	798	>1800	**798**	522	1168.5
t2-ps02	**784**	157.8	**784**	88.4	**784**	7.7	14.8
t2-ps03	**749**	1897.7	**749**	144.2	**749**	47.8	111.7
t2-ps04	**730**	188.8	**730**	388.3	**730**	34.3	93.6
t2-ps05	**691**	770.1	**691**	30.4	**691**	30.2	136.7

Table 4 compare the results of our approach with BT on the $5 \times 15 \times 5$ instances. The results of Focacci *et al* [9] are not available on these instances. We also indicate the current value of the lower bound when the algorithm is stopped, for our approach. For $5 \times 15 \times 5$ instances, all the results from Brucker and Thiele [6] are improved. Two new instances of this size are closed for the first time (t2-ps10 and t2-ps07), with additional CPU time requirements for the latter one. A comparison with BT is here even more difficult for two reasons. First, it would be necessary to test the BT Branch and Bound on a recent computer. Second, as shown in table 2 all upper bounds given in a few seconds by the fast heuristic proposed in [2] are better than the ones provided by BT after 2 hours of computation. Hence, a better heuristic should be used before running the BT algorithm.

Table 4. Makespan and lower bound value for instances $5 \times 15 \times 5$

Instance	BT		ABF		
	Cmax	CPU (sec.)	Cmax	BInf	CPU (sec.)
t2-ps06	1056	7200	1026	996	7200
t2-ps07	1087	7200	970 (**970**)	966 (970)	7200 (16650)
t2-ps08	1096	7200	1002	923	7200
t2-ps09	1119	7200	1060	1037	7200
t2-ps10	1058	7200	**1018**	1018	498

To compare our approach on large instances with the one of Brucker and Thiele [6] and Focacci *et al* [9], we give in table 5 the results on the two SDST-JSP $5 \times 20 \times 10$ used in [9]. The limits of our approach is reached since after 2 hours the dichotomizing search is unable to increase the initial lower bound displayed in column ABF/LB and does not improve the initial upper bound displayed in column ABF/Cmax. However the initial upper bound is here again surprisingly better than the results obtained by FNL and BT and the proposed lower bound still improves the BT lower bound.

Table 5. Makespan and lower bound value for instances $5 \times 20 \times 10$

Instance	BT			FNL		ABF		
	Cmax	CPU (sec.)	LB	Cmax	CPU (sec.)	Cmax	CPU (sec.)	LB
t2-ps12	1528	7200	1139	1448	120	1319	7200	1159
t2-ps13	1549	7200	1250	1658	120	1439	7200	1250

6 Conclusion

In this paper, we propose a new algorithm for the exact solution of the SDST-JSP. The main framework of the algorithm is a Branch and Bound strategy embedding constraint propagation at each node of the search tree. In order to prune nodes effectively, we introduce a global constraint by relaxing precedence constraints between operations. Filtering algorithm for this global constraint reduces to solving a TSPTW instance for each machine. The principle is to detect infeasible TSPTW instances, indicating unfeasibility for the global problem associated to the current node of the search tree. In order to improve computing time, we use a long term memory by storing feasible solutions of TSPTW instances found so far. This permits to limit significantly the negative impact of the global constraint on computing time.

This approach is competitive with the approaches of Brucker and Thiele [6] and Focacci *et al* [9] and permits to close some new instances. Possible improvements in the future are the introduction of a Lagrangian relaxation type penalization of the precedence constraints relaxed within the bounding scheme and the implementation of more efficient algorithms for the solution of TSPTW instances (as the ones proposed in Pesant *et al.* [14] or Focacci and Lodi [10]).

Acknowledgements

The authors would like to thank Sophie Demassey for her help as a LaTex specialist. We would like also to thank the anonymous referees for their helpful comments.

References

1. Adams J., Balas E. and Zawack D. (1988) The Shifting Bottleneck Procedure for Job-Shop Scheduling. Manag Sci 34:391-401.
2. Artigues C., Lopez P. and Ayache P.D. (2003), Schedule generation schemes and priority rules for the job-shop problem with sequence dependent setup times: Dominance properties and computational analysis. to appear in Annals of Operations Research.
3. Artigues C. and Buscaylet F. (2003), A fast tabu search method for the job-shop problem with sequence-dependent setup times. Proceedings of the Metaheuristic International Conference MIC'2003, Kyoto 2003.
4. Baptiste P., Le Pape C. and Nuijten W. (2001), Constrained-Based scheduling, Applying constraint programming to Scheduling Problems, Kluwer's International Series, 19-37.
5. Blazewicz J. ,Domschke W. and Pesch E. (1996), The job shop scheduling problem: Conventional and new solution techniques. European Journal of Operational Research, 93(1):1–33.

6. Brucker P. and Thiele O. (1996), A branch and bound method for the general-shop problem with sequence-dependent setup times, Operations Research Spektrum, 18, 145-161.
7. Carlier J. and Pinson E. (1989), An algorithm for solving the Job-Shop Problem. Management Science 35, 164-176
8. Choi I.-N. and Choi D.-S. (2002), A local search algorithm for jobshop scheduling problems with alternative operations ans sequence-dependent setups, Computers and Industrial Engineering 42, 43-58.
9. Focacci F., Laborie P. and Nuijten W. (2000), Solving scheduling problems with setup times and alternative resources. In Fifth International Conference on Artificial Intelligence Planning and Scheduling, Breckenbridge, Colorado, 92-101.
10. Focacci F. and Lodi, A. (1999), Solving TSP with Time Windows with Constraints, International Conference on Logic Programming 515-529
11. ILOG SOLVER 5.2 User's Manuel (2000), ILOG
12. ILOG SCHEDULER 5.2 User's Manuel (2000), ILOG
13. Nowicki E. and Smutnicki C. (1996), A fast taboo search algorithm for the job shop problem, Management Science 42, 797-813.
14. Pesant G., Gendreau M., Potvin J-Y. and Rousseau J-M. (1999), An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows, European Journal of Operational Research 117, 253-263.
15. Vaessens R.J.M., Aarts E.H.L. and Lenstra J.K. (1996), Job Shop Scheduling by Local Search, INFORMS Journal on Computing 8, 302-317.
16. Vilm P. and Bartk, R. (2002), Filtering Algorithms for Batch Processing with Sequence Dependent Setup Times, In Ghallab, Hertzberg, and Traverso (eds.) Proceedings of The Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002). AAAI Press, Toulouse, p. 312-320.