# A Constraint-based Job-Shop Scheduling Model for Software Development Planning

Irene Barba, Carmelo Del Valle, and Diana Borrego

Dpto. Lenguajes y Sistemas Informáticos,
Universidad de Sevilla, Spain
{irenebr,carmelo,dianabn}@us.es

**Abstract.** This paper proposes a constraint-based model for the Job Shop Scheduling Problem to be solved using local search techniques. The model can be used to represent a multiple software process planning problem when the different (activities of) projects compete for limited staff. The main aspects of the model are: the use of integer variables which represent the relative order of the operations to be scheduled, and two global constraints, *alldifferent* and *increasing*, for ensuring feasibility. An interesting property of the model is that cycle detection in the schedules is implicit in the satisfaction of the constraints. In order to test the proposed model, a parameterized local search algorithm has been used, with a neighborhood similar to the Nowicki and Smutnicki one, which has been adapted in order to be suitable for the proposed model.

**Key words:** job shop scheduling, local search, constraint satisfaction problems, software development processes

## 1 Introduction

Software development has been modelled using a wide range of approaches. They vary according to the focus of the analysis and they address successfully the whole development process depending on how it is carried out. Many of the software management tools use temporal information and ignore in some ways the resources to be used, considering them unlimited, since they are based on PERT and CPM analysis. These may not be adequate in different situations, for example when smaller multiple projects are developed and project compete for limited staff [10]. A job shop approach, traditional in manufacturing, may represent an important aid since it can manage the interactions between projects and resources in a natural way and enables to consider minimizing different goals, as development time (makespan) and cost, while satisfying all the temporal and resource constraints.

With this aim, a job shop scheduling model is presented in this paper, so that it can represent a multiple software project to be planned. The equivalence of terms used from both areas is in such a way that jobs correspond to single software projects, and resources can represent each person or software development team working in the projects.

Constraint Programming (CP) has evolved in the last decade to a mature field due to, among others, the use of different generic and interchangeable procedures for inference and search, which can be used for solving different types of problems [2, 12]. Although a separation of models and algorithms is desirable for reusability issues, there is an influence between them that must be taken into account when a good behavior of the whole resulting method is pursued. Most models that have been used in CP have been tested using complete algorithms, and they are not equally suitable for other algorithmic approaches such as local search [16].

This paper proposes a Constraint Satisfaction Problem (CSP) model for the Job Shop Scheduling Problem (JSSP) to be solved using local search techniques, that is, it defines the variables which determine a solution, the related constraints of the problem involving those variables, and some possible neighborhoods. The problem has been solved by different authors using local search [8, 11, 15], but the novelty consist of the proposed model, based on including the ordering of the operations directly in the variables and constraints of the CSP, so that further definitions and developments of the main components of local search algorithms would take advantage of this representation.

For such techniques, a very important issue is the defined neighborhood, that is, the set of candidates to which the walk may continue from the current solution. For JSSP, one of the best methods was proposed by Nowicki and Smutnicki [11], whose neighborhood was more constrained than other previous approaches. An adaptation and an extension of this neighborhood are proposed in this work, in order to be suitable for the defined CSP model.

The rest of the paper is organized as follows. Section 2 presents a formulation of the JSSP. Section 3 includes the main ideas of local search algorithms. Section 4 describes the proposed model. Next, experimental results are shown and analyzed. Finally, Section 6 presents some conclusions and future work.

## 2 Problem Definition

The Job Shop Scheduling Problem [1, 8] may be formulated as follows. We are given a set of $n$ jobs $J_1, ..., J_n$ and a set of $m$ machines $M_1, ..., M_m$. Each job $J_i$ consists of a sequence of $n_i$ operations $op_{i1}, ..., op_{i,n_i}$, which must be processed in this order. Each operation $op_{ij}$ must be processed for $p_{ij}$ time units, without preemption, on machine $\mu_{ij} \in \{M_1, ..., M_m\}$. Each machine can only process one operation at a time. So, two types of constraints are defined, the precedence constraints among the operations of each job, and the resource constraints which force to select a permutation order of the operations that use each machine. These last constraints are the source of the NP-hard complexity of JSSP [4].

The typical objective, used in this work, is to find a feasible solution, minimizing the makespan, $C_{max} = max_{i=1..n}\{C_i\}$, where $C_i$ is the completion time of job $J_i$, i.e. the completion time of $op_{i,n_i}$.

Figure 1 shows the disjunctive graph representation for a simple example of the problem, with $n = 3$ and $n_i = 3$, $\forall i$. In a disjunctive graph $G = (V, C, D)$, we have a set $V$ of nodes which correspond to the operations of the job-shop, a set $C$ of directed arcs corresponding to the precedence constraints, and a set $D$ of undirected arcs which connect the operations that use the same machine. A solution to the problem consists of fixing a direction for the undirected arcs, being feasible if there are no cycles.

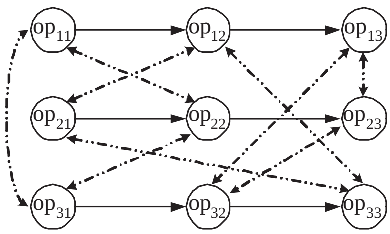


**Fig. 1.** A disjunctive graph for a job shop problem.

## 3 Constraint-Based Local Search

Most solving algorithms for CSPs proposed in the CP area are complete, and lastly local search is being considered as promising for solving large instances of complex problems [16], where complete algorithms fail. The constraints from the CSP model may be used for guarantying feasibility of the solutions explored, or even using their possible (degree of) violation as a guide for the search. Most of ideas associated to local search algorithms in other areas can be used for solving CSPs, or in our case, a Constraint Optimization Problem (COP).

Local search algorithms move iteratively through the set of feasible solutions. For those movements, a neighborhood for the current solution is determined in each iteration as a set of the solutions that can be selected as the next solution, and that can be obtained from the current solution with small changes. Depending on the method of choosing the next solution from neighborhood and the criteria for stopping the iterative sequence of movements, different algorithms can be defined [6]. In order to test the proposed model, we have used a basic tabu search algorithm [5] containing the main components that have been proved useful in local search, as described in Section 4.4.

## 4 Our Proposal

### 4.1 The CSP Model

A CSP is defined by a set of variables $V$, a set of domains of values for each variable $D$ and a set of constraints that involve the variables $C$. Typical CSP models for the JSSP state the start times $st_{ij}$ of the operations $op_{ij}$ as the variables of the CSP [3], and the constraints are divided in two groups, precedence constraints

($st_{ij} + p_{ij} \leq st_{i,j+1}$) and resource constraints ($st_{ij} + p_{ij} \leq st_{kl} \vee st_{kl} + p_{kl} \leq st_{ij}$, $op_{ij}$ and $op_{kl}$ using the same machine). Our proposed CSP model is based on using the CSP variables to establish the execution orden of the operations of the JSSP, resulting in a simple model.

Let $\Pi_J$ be a JSSP with a set $J$ of $n$ jobs, a set $M$ of $m$ machines, and a set $O$ of $\#ops$ operations. The proposed model has the following components:

- Each operation $op_{ij}$ is represented as an integer variable of the CSP $v_{ij}$, therefore the set of variables is $V = \{v_{ij}, 1 \leq i \leq n, 1 \leq j \leq n_i\}$.
- The domain of each variable $v_{ij}$ is $D(v_{ij}) = [1..\#ops], \forall v_{ij} \in V$.
- The set $C$ of constraints contains two types of items:
  1. **Precedence Constraints**: The value of each variable $v_{ij}$ has to be less than the value of all the variables corresponding to the following operations in the same job: $v_{ij} < v_{ik}, \forall v_{ij}, v_{ik}$ such that $j < k$. In order to improve the efficiency and to obtain a clearer model, a new constraint (*increasing*) has been used between the operations of each job. It is defined on a sequence of variables $\{v_1, v_2, \ldots, v_n\}$ and it is equivalent to the satisfaction of the conditions $v_1 < v_2 < \ldots < v_n$.
  2. **Resource Constraints**: In order to satisfy that each machine can process only one operation at the same time, all the variable values are forced to be different from the others (*alldifferent* constraint is used), i.e., each solution is a permutation of the set $\{1, 2, \ldots, \#ops\}$.

An interesting property of the model, using the *increasing* and *alldifferent* constraints, is that cycle detection in the disjunctive graph is implicit in the satisfaction of the constraints, so no solution of the CSP will contain cycles.

A solution for the constrained problem, in which a value for each CSP variable is given, is a permutation of $1..\#ops$ variables and can be represented by an ordered sequence of operations $S$. With this sequence we associate an "earliest start schedule" by planning the operations in the order induced by the sequence, resulting in a JSSP solution. We denote $S(m)$ as the ordered sequence of operations that are executed on the machine $m$ in the order fixed by the solution represented by $S$. Figure 2 shows a solution for the problem of Fig. 1. First, the value for each variable is shown and below is the corresponding solution $S$, where the position $a$ in the sequence represents the value of the variable $S[a]$ ($v_{ij} = a \equiv S[a] = op_{ij}$). Also, the ordered sequences corresponding to each machine are shown. Finally, Fig. 2 shows a JSSP solution where all the arcs in the graph are directed according to the fixed order in $S$. Notice that there can be several solutions of the CSP problem that lead to the same schedule, for example the solution $S = \{op_{21}, op_{31}, op_{32}, op_{11}, op_{12}, op_{13}, op_{22}, op_{33}, op_{23}\}$ for the problem of Fig. 1 leads to the same schedule that the solution shown in Fig. 2.

From now, we will use $PM(v)$ and $SM(v)$ to refer to the predecessor and successor variables of $v$ on its machine, and similarly $PJ(v)$ and $SJ(v)$ on its job. $PM(PM(v))$ is denoted by $PM_2(v)$ (the same for $SM(v)$) and so on. Moreover, we denote $m(v)$ as the machine in which the operation corresponding to the variable $v$ has to be executed.

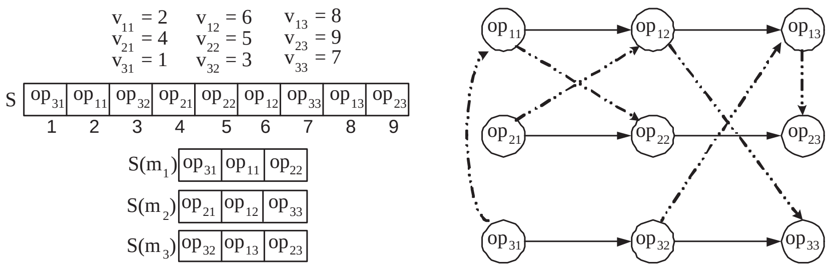


**Fig. 2.** Example of a feasible solution

### 4.2 Cycle Detection

A solution for the problem consists of establishing directions for the undirected arcs in the disjunctive graph (Section 2), being feasible if there not exists any cycle. A cycle for a solution in the disjunctive graph is a closed directed (simple) path, with no repeated vertices other than the starting and ending vertices.

We can see a cycle as a sequence of operations that contains two types of edges:

- Precedence edges: are fixed by the problem.
- Resource edges: are given by the decisions made to solve the problem.

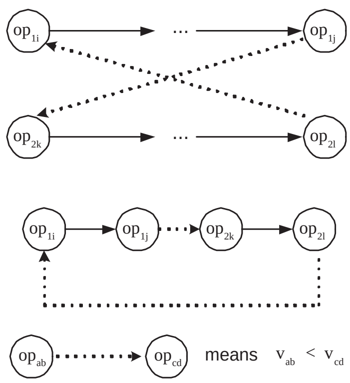


**Fig. 3.** A cycle in a disjunctive graph

All the possible cycles that can be formed in the graph involve, at least, two machines and four operations, two belonging to one job, and two belonging to another job, such as it is shown in the figure 3. In this figure it is possible to see a cycle formed by four operations, two belonging to $J_1$ ($op_{1i}$ and $op_{1j}$) and two belonging to $J_2$ ($op_{2k}$ and $op_{2l}$). In the sequence of operations appears, at least, two precedences edges, that connect operations using different machines. All the operations that appear in the figure can be executed on different machines, so the machines involved in this cycle can be between 2 and 4. It is important to clarify that $op_{1,j}$ and $op_{2,k}$ do not have to be executed in the same machine (the same for $op_{1,i}$ and $op_{2,l}$).

It can be proven that any solution of the CSP, with the proposed model, will contain no cycles.

### 4.3 Neighborhoods

For JSSP, most of the successful approaches use neighborhood based on reversing critical operations (increasing their durations imply a larger makespan) that must be processed on the same machine. One of the best methods was proposed by Nowicki and Smutnicki [11], whose neighborhood was more constrained than other previous approaches. The movements allowed were to reverse two adjacent critical operations belonging to the same critical block (a sequence of critical operations on the same machine) so that one of them is not an internal operation in the block, excluding the swap between the first two operations of the first block when the second one is internal, and the swap between the last two operations of the last block when the first is internal.

We define a family of neighborhoods for the proposed model in which the basic idea is to make a swap between the values of two variables corresponding to operations of the same machine, i.e., between the relative order of those operations in a solution, trying to change the order of operations belonging to a critical path of a solution $S$ ($CP(S)$ from now), based on the Nowicki and Smutnicki (NS from now) neighborhood.

For a variable $v$, $\sigma(v)$ is defined as the set of the variables $w$ satisfying the following condition: **the swap between $v$ and $w$ in $S$** (denoted as $swap(v, w, S)$) **causes a swap between $v$ and $PM(v)$ on $m(v)$ and this is the only swap caused on $m(v)$**. The variables $w$ that meet this condition are those between $PM_2(v)$ (not included) and $PM(v)$ (included) in $S$. We can see that the swaps between $v$ and variables that appear before $PJ(v)$ in $S$ lead to unsatisfiable solutions. Then, $\sigma(v)$, when $v$ is not the first in its job and has, at least, two predecessors on its machine, is defined as:

$$\sigma(v) = \{w \in V \mid max(PJ(v), PM_2(v)) < w \leq PM(v)\}$$

If $PJ(v)$ and $PM_2(v)$ do not exist, the outer lower bound is 0. On the other hand, if only one of them exists, the outer lower bound is established by it. Lastly, all the variables which have the smallest value on their machine (i.e., which are executed first) do not have any possible swaps ($\sigma = \emptyset$).

In Fig. 4 different cases of possible swaps are shown. In Fig. 4.a, $PJ(v)$ appears before $PM_2(v)$, then the outer lower bound of the range of possibilities is established by $PM_2(v)$. In Fig. 4.b, $PM_2(v)$ is before $PJ(v)$, then it is given by $PJ(v)$. In Fig. 4.c, $PJ(v)$ is after $PM(v)$, so no swap for $v$ can be realized.

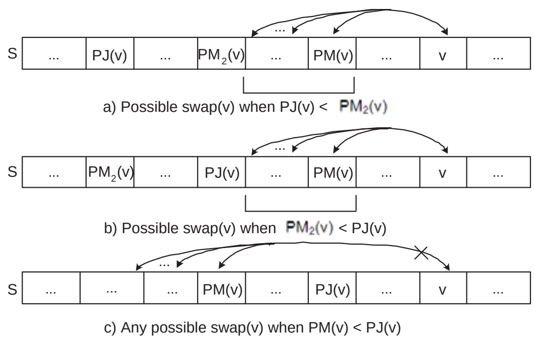


**Fig. 4.** Possible swaps for a variable $v$

In order to reduce and set a maximum number of neighbors for a solution, we define a parameter $\delta$, as the maximum number of possible swaps for a variable $v$, from $PM(v)$ toward variables appearing before it in $S$. It must be noticed that $\delta$ has to be greater than 1 so that the algorithm can reach any possible solution, taking into account the proposed model. According to this parameter, the set of considered swaps for a variable, is defined as:

$$\sigma_\delta(v) = \{w \in V \mid max(PM(v) - \delta, PJ(v), PM_2(v)) < w \ \wedge \ w \leq PM(v)\}$$

A family of neighborhoods, $N^\delta_{1\lambda}$, depending on the possible variables to swap, has been defined. For $\lambda = 0$, the idea is to swap variables that are at the beginning or at the end of a critical block (CB from now, $CB(v)$ for the CB of a variable $v$), except the beginning of the first CB or the end of the last CB, similar to NS neighborhood. These variables are given by the set $V_0(S)$:

$$V_0(S) = \{v \in CP(S) \mid v = SM(first(CB(v))) \vee v = last(CB(v))\}$$

where $first(CB(v))$ and $last(CB(v))$ are the first and the last operations of $CB(v)$, respectively. $V_0(S)$ contains the possible variables to be swapped in $N^\delta_{10}$ ($N^\delta_{10} = \{swap(v, w, S) \mid v \in V_0(S) \wedge w \in \sigma_\delta(v)\}$).

Due to the proposed model and the tabu search, it is possible to reach a solution which an empty neighborhood. In order to overcome this problem and get more diversification during the search, other more general neighborhoods $N^\delta_{1\lambda}$, different from NS proposal, have been defined depending on a parameter $\lambda$. For $\lambda > 0$, it is allowed to swap internal variables of CBs, more internal as $\lambda$ is increasing. The set of possible variables to swap, is now given by:

$$V_\lambda(S) = \{v \in CP(S) \mid (v = SM_{\lambda+1}(first(CB(v))) \vee v = PM_\lambda(last(CB(v)))) \wedge \lambda \leq \#CB(v)/2\}$$

Then, the neighborhood $N^\delta_{1\lambda}$ is defined as $N^\delta_{1\lambda} = \{swap(v, w, S) \mid v \in V_\lambda(S) \wedge w \in \sigma_\delta(v)\}$. In order to allow swaps between all the non-critical operations (belonging or not to CP), another neighborhood has been defined: $N^\delta_2 = \{swap(v, w, S) \mid v \in V \wedge w \in \sigma_\delta(v)\}$.

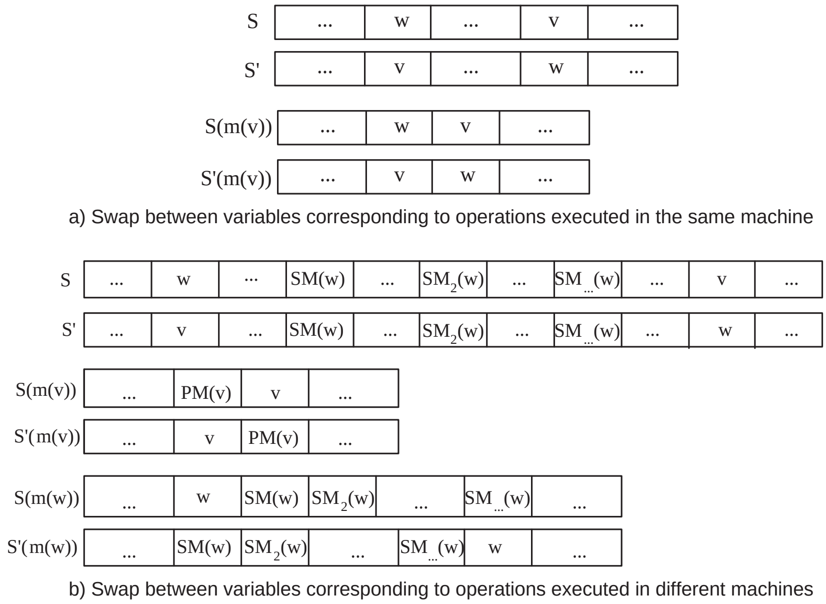


**Fig. 5.** Swap between variables

The swap between $v$ and $w$ has the following consequences:

1. Swap between the execution order of $v$ and $PM(v)$, executed on the same machine, **which does not depend on** $w$ (change in $m(v)$).
2. If $w \neq PM(v)$, i.e. $m(w) \neq m(v)$, other changes will be given. If $SM(w) < v$, then the execution orders of all the operations $w'$ satisfying $m(w') = m(w)$ and $w < w' < v$ will be changed. Specifically, the relative order of all these operations are moved forward on their machine (change in $m(w)$).

According to this, two types of movements can be given. First, the swap between variables corresponding to operations executed on the same machine, only one swap in $S(m(v))$ is given (Fig. 5.a). Secondly, the swap between variables corresponding to operations executed on different machines, that leads to a swap in $S(m(v))$ and several swaps in $S(m(w))$, one for each direct or indirect successor on the machine of $w$ that is between $w$ and $v$ in $S$ (Fig. 5.b). In Fig. 5 the neighbor for $S$ is referred as $S'$.

### 4.4 The Parameterized Algorithm

Considering the defined neighborhoods, a local search algorithm has been developed (Alg. 1). Although any initial solution can be used, the choice of better initial solutions usually allows to obtain better results, as it is found for the NS method [9]. In this way, for the experiments of the next section, we have

**Algorithm 1**: The parameterized local search algorithm

```
begin
    determine an initial solution S randomly
    best := S;
    for it := 1 to NIterations do
        if solution has not improved in last K iterations then
            choose a neighbor S' of best in N_2^δ randomly;
        else
            λ := 0;
            repeat
                determine set N_{1λ}^δ of non-tabu neighbors of S;
                if N_{1λ}^δ is not empty then
                    choose a best solution S' in N_{1λ}^δ;
                else
                    λ := λ + 1;
            until S' has been selected or λ > maxBlockSize(S)/2 ;
            if S' has not been selected (all of N_{1λ}^δ are empty) then
                choose a neighbor S' of S in N_2^δ randomly;
        S := S';
        if S improves best then
            best := S;
end
```

used the INSA algorithm [11]. As indicated in Subsection 4.1, a schedule can be represented by different solutions of the model. Thereby, for selecting the actual initial solution a random procedure is used from the schedule obtained by the INSA algorithm.

According to the evolution of the search, different neighborhoods are used in order to select the next movement, which will correspond to a feasible solution. In each iteration, a movement to the best neighbor of $N^{\delta}_{1,0}$ is attempted ($\lambda = 0$), but, if the neighborhood is empty or all their members are in the tabu list, a more extended neighborhood is searched, by increasing $\lambda$. If $\lambda$ reaches the allowed maximum value without finding a suitable next solution to visit, the more general neighborhood $N^{\delta}_{2}$ is used, and now the neighbor is selected randomly. $N^{\delta}_{2}$ is also used when the algorithm has not found a better solution for a number $K$ of iterations. In this case, the algorithm returns to the best solution found so far.

Besides that, most of the computational cost of local search algorithms are due to the evaluation of neighbors. In order to reduce its amount, several approaches have been proposed, such as that of Taillard [14], which evaluates the neighbors using a lower bound estimation of the makespan in constant time, instead of calculating it exactly. In the proposed algorithm, the selection of candidates is made in two steps. First, the best swap between two critical operations is selected using the Taillard estimation of the expected makespan . After that, a variable is selected from the $\delta$ possibilities, choosing the one with the greatest improvement in its slack because of the change.

## 5 Experimental Results

ILOG JSolver [7] has been used for implementing the Algorithm 1, and for managing the constraints of the problem. As stated before, the algorithm has several parameters, $\delta$, $K$ (maximum number of iterations without improving the solution), and the tabu list size ($TLS$), that may affect its behavior, and its tuning represents a non-trivial problem. Since the main interest of this work is not the competitiveness of the algorithm proposed, but the CSP model which is defined, a scenario for some comparative results was chosen, in which the algorithm would be executed for a fixed number of 10000 iterations and , which were selected randomly from the results of the INSA algorithm. For such situation, the value of $K$ was chosen to be 1000. For selecting $\delta$ and $TLS$, the algorithm was run on a reduced set of instances for $\delta$ from 2 to 5 and from $TLS$ from 5 to 10. The best results on the minimal and average makespan of the best solution after 10000 iterations were found for $\delta = 2$ and $TLS = 6$. The best results for $\delta = 2$ can be explained by the fact that for higher values of $\delta$, there is more probability for finding a variable $w$ such that the swap between $v$ and $w$ will be feasible, which would enforce the diversification strategy too much.

Table 1 shows the results of the algorithm for a larger set of JSSP benchmarks, taken from the OR-library, and some harder instances from Taillard [13]. For each JSSP instance, the table shows some statistics about the algorithm used in this work: the relative error of the best solution from the 100 restarts ($BRE\%$) with respect to the best known solution ($UB$, which is not proved optimal for the values indicated by *), the mean relative error ($MRE\%$) and the standard deviation of relative error ($SDRE\%$). Also, the mean computational time for running the algorithm is given ($RT$). As reference, the results obtained by the NS algorithm is shown in two situations: in the original form, that takes into account several factors, and after 10000 iterations. As expected, the algorithm is not fully competitive (as well as it has been developed in Java, many of its components are not optimized) with that of Nowicki and Smutnicki, considered as one of the best methods for solving the JSSP. Instead, the results shown must be taken as a reference for further improvements of the algorithm or for different approaches that can use the model.

## 6 Conclusions and Future Work

This paper proposes a CSP model for the Job Shop Scheduling Problem to be solved using local search techniques. The model can be used to represent a multiple software process planning problem when the different (activities of) projects compete for limited staff. The main aspects of the model are the use of integer variables which represent the relative order of the operations to be scheduled and two types of global constraints for ensuring feasibility. Also, a

**Table 1.** Results on a set of JSS instances

| | | | | Proposed Model | | | | NS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Instance* | *n* | *m* | *UB* | *BRE%* | *MRE%* | *SDRE%* | *RT* | *BRE%* | *RT* | $BRE\%_{10^4}$ | $RT_{10^4}$ |
| FT10 | 10 | 10 | 930 | 2.25 | 3.95 | 0.96 | 8.72 | 0 | 0.68 | 0 | 0.25 |
| ABZ7 | 20 | 15 | 656 | 9.90 | 13.98 | 2.16 | 64.40 | 2.28 | 4.62 | 3.20 | 0.84 |
| LA02 | 10 | 5 | 655 | 0.45 | 3.80 | 1.83 | 3.02 | 0 | 0.10 | 0 | 0.11 |
| LA19 | 10 | 10 | 842 | 2.49 | 6.25 | 1.82 | 10.52 | 0.11 | 0.83 | 0.11 | 0.35 |
| LA21 | 15 | 10 | 1046 | 5.16 | 8.23 | 1.05 | 18.76 | 0.86 | 0.86 | 0.86 | 0.42 |
| LA24 | 15 | 10 | 935 | 3.85 | 6.56 | 1.13 | 18.72 | 1.39 | 1.33 | 1.50 | 0.45 |
| LA25 | 15 | 10 | 977 | 7.26 | 11.24 | 1.84 | 20.65 | 1.12 | 1.39 | 2.04 | 0.45 |
| LA27 | 20 | 10 | 1235 | 6.96 | 12.07 | 2.03 | 30.73 | 1.94 | 1.27 | 1.94 | 0.51 |
| LA29 | 20 | 10 | 1152 | 8.42 | 12.57 | 2.22 | 33.24 | 3.13 | 3.40 | 4.51 | 0.48 |
| LA36 | 15 | 15 | 1268 | 7.09 | 11.42 | 1.25 | 38.61 | 0.79 | 3.66 | 2.76 | 0.62 |
| LA37 | 15 | 15 | 1397 | 9.09 | 14.59 | 2.30 | 41.80 | 1.50 | 2.74 | 3.29 | 0.78 |
| LA38 | 15 | 15 | 1196 | 5.85 | 8.09 | 0.99 | 41.72 | 1.84 | 2.75 | 2.59 | 0.65 |
| LA39 | 15 | 15 | 1233 | 7.94 | 10.44 | 0.90 | 41.04 | 0.89 | 3.50 | 1.62 | 0.79 |
| LA40 | 15 | 15 | 1222 | 7.03 | 10.28 | 1.02 | 36.52 | 1.64 | 2.40 | 2.13 | 0.62 |
| TA02 | 15 | 15 | 1244 | 6.35 | 10.49 | 1.64 | 36.47 | 2.73 | 2.83 | 2.73 | 0.70 |
| TA18 | 20 | 15 | 1396* | 12.60 | 15.25 | 1.32 | 57.82 | 3.65 | 4.64 | 5.73 | 0.97 |
| TA26 | 20 | 20 | 1645* | 9.36 | 13.14 | 1.34 | 93.66 | 3.10 | 10.64 | 3.28 | 1.58 |
| TA32 | 30 | 15 | 1795* | 14.20 | 17.84 | 1.30 | 116.93 | 3.12 | 18.36 | 6.85 | 1.44 |

neighborhood for this model has been defined based on an adaptation of Nowicki and Smutnicki one. The main focus is not on the competitiveness of the algorithm which is proposed, but in the definition of the CSP model.

As future work, the algorithm and neighborhood should be improved for solving more efficiently the JSSP. Also, we think that the proposed model can be adapted for other similar sequencing problems in a direct way.

On the other hand, it is intended to extend the software development process to other (more generic or specific) models and to adapt the corresponding (planning and/or) scheduling models and solving algorithms.

**Acknowledgments.** This work has been partially supported by the Spanish Ministerio de Educación y Ciencia through a coordinated research project(grant DIP2006-15476-C02-01) and Feder (ERDF).

## References

1. P. Brucker and S. Hnust, *Complex Scheduling*, Springer, 2006.
2. R. Dechter, *Constraint Processing*, Morgan Kaufmann Publishers, 2003.
3. R. Dechter, I. Meiri, and J. Pearl, 'Temporal constraint networks', *Artificial Intelligence*, **49**, 61–95, (1991).

4. M. R. Garey, D. S. Johnson, and R. Sethi, 'The complexity of flowshop and jobshop scheduling', *Math. Oper. Res.*, **1(2)**, 117–129, (1976).
5. F. Glover and M. Laguna, *Tabu Search*, Blackwell Scientific Publishing, Oxford, England, 1993.
6. H. H. Hoos and T. Stutzle, *Stochastic Local Search. Foundations and Applications*, Morgan Kaufmann, 2005.
7. ILOG, 'Ilog JSolver', (2003).
8. A. Jain and S. Meeran, 'Deterministic job-shop scheduling: Past, present, and future', *European Journal of Operational Research*, **113 (2)**, 390–434, (1999).
9. A. Jain, B. Rangaswamy, and S. Meeran, 'New and Stronger Job-Shop Neighbourhoods: A Focus on the Method of Nowicki and Smutnicki (1996)', *Journal of Heuristics*, **6**, 457-480, (2000).
10. C. A. Leonhard, and J. S. Davis, 'Job-Shop Development Model: A Case Study', *IEEE Software*, **12 (2)**, 86-92, (1995).
11. E. Nowicki and C. Smutnicki, 'A fast taboo search algorithm for the job-shop problem', *Management Science*, **42(6)**, 797813, (1996).
12. F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, Elsevier, 2006.
13. E. Taillard, 'Benchmarks for basic scheduling problems', *European Journal of Operational Research*, **64**, 278-285, (1993).
14. E. Taillard, 'Parallel Taboo Search Techniques for the Job-Shop Scheduling Problem', *ORSA Journal on Computing*, **16(2)**, 108-117, (1994).
15. R. Vaessens, E. Aarts, and J. Lenstra, 'Job-shop scheduling by local search', *INFORMS Journal on Computing*, **8**, 302–317, (1994).
16. P. Van Hentenryck and L. Michel, *Constraint-Based Local Search*, The MIT Press, 2005.