Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs

Emilie Danna[1,2] and Laurent Perron[2]

Laboratoire d'Informatique d'Avignon[1]
339, chemin des Meinajariès, Agroparc, BP 1228
F-84911 Avignon Cedex 9

ILOG SA[2]
9 rue de Verdun
F-94253 Gentilly Cedex

{edanna,lperron}@ilog.fr

Abstract

Large Neighborhood Search (LNS) is a local search paradigm that defines larger neighborhoods than typical local search algorithms, and explores them with powerful algorithms usually based on tree search — constraint programming or mixed integer programming (MIP). Even more than in any local search algorithm, the key question in LNS is how to define neighborhoods. We compare in this paper a *structured* LNS approach embedded in constraint programming that exploits the known high level structure of the specific problem at hand to define its neighborhoods, and a recent *unstructured* LNS approach called Relaxation Induced Neighborhood Search that relies only on the continuous relaxation to define its neighborhoods, hence that can be used on any MIP model with no other input than the model itself. We show that both our LNS approaches outperform a variety of existing algorithms on several difficult benchmarks for the job-shop scheduling problems with earliness and tardiness costs. Our structured LNS approach dramatically improves on pure constraint programming because it defines a cost structured neighborhood that is able to attack a sum objective, on which constraint propagation was traditionally very weak.

1 Introduction

When defining a local search algorithm, choosing the right neighborhood size is a difficult question to answer. Defining too small a neighborhood will hinder the algorithm's ability to escape local search optima. In turn, defining too large a neighborhood will slow down the choice of a suitable move in the neighborhood, its evaluation and its realization.

Large Neighborhood Search (LNS) [26] is a local search paradigm based on two main ideas to define and search large neighborhoods. The first key idea of LNS is to define its neighborhoods by fixing or *freezing* a part of an existing solution. The elements of the solution that are fixed are usually explicit or implicit variables of the model. For example, in a scheduling model, one may choose to fix the values of the start times of each activity (explicit variables) or one may add additional constraints that force one activity to be scheduled before another (implicit disjunctive variables). The rest of the variables are *released*: they are free to change values. The neighborhood is hence defined by all possible extensions of the fixed partial solution. Because a number of variables are released at a time, the neighborhoods defined are usually large, larger than typical local search neighborhoods.

The size of the so-defined neighborhood requires a powerful algorithm to search it; one cannot rely on enumeration or simple heuristics to find interesting moves in it. The second key idea of LNS is to use some form of tree search, constraint programming (CP) or mixed integer programming (MIP) to search its neighborhoods, *i.e.*, to solve its sub-problems. The tree search is most often truncated with a possibly adaptive time limit, node limit or discrepancy limit. Note that LNS

is often recursive, the same algorithm being used to generate a first solution and to solve the sub-problems in a second stage.

The large size of the neighborhoods and the powerful algorithm used to solve sub-problems provide LNS with inherent diversification and intensification properties, respectively. Therefore the essential question in LNS, even more than in a typical local search algorithm, is how to define the neighborhoods, that is which variables to choose and fix together. The rule to define a promising neighborhood is to free simultaneously *related* variables. First, the problem might be so constrained that, even when freeing a number of randomly chosen variables, there might exist no other extension of the so-defined partial solution than the current solution. It is therefore essential to free related variables because they allow each other to change values. Next, the essence of LNS is to compute more complex moves that yield a new solution further away from the current solution than when exploring smaller neighborhoods. This provides diversification and this allows us to solve difficult core sub-problems. But LNS will succeed only if the neighborhood it defines corresponds to a sub-problem that is not a concatenation of smaller and independent sub-problems but a consistent core problem in itself. One hopes indeed that the gain obtained by simultaneously computing new values for the released variables will be greater than the gain obtained by changing the value of each variable independently. It is therefore essential to free related variables because they allow each other to take more meanigful values.

In the literature, choosing related variables is most often achieved by taking advantage of the known *high level structure* of the specific problem at hand: the neighborhoods are carefully tailored to capture a good approximation of the relatedness [26] function between the variables. Recently, a new algorithm called *Relaxation Induced Neighborhood Search* (RINS) [10] was introduced: it is a form of LNS that only relies on the continuous relaxation of the MIP model of the problem to define its neighborhood. It can be used on any MIP model without any other input than the model itself. Unlike all previous LNS algorithms, it is therefore *unstructured*.

Structured Large Neighborhood Search methods have been proved to be very successful on a variety of hard combinatorial problems, *e.g.*, vehicle routing [26, 6, 8], scheduling (job-shop: shuffle [1], shifting bottleneck [17], forget-and-extend [7]; RCPSP: block neighborhood [21]), network design [18, 9], frequency allocation [20]. In turn, RINS has shown to improve dramatically the ability of MIP solvers to solve various very difficult MIP models, outperforming even novel MIP heuristics based on similar ideas integrating MIP and local search [10, 11].

The aim of this paper is to compare relaxation induced neighborhood search and a structured large neighborhood search approach tailored to a particular and difficult problem. On the one hand, we want to evaluate how powerful LNS approaches are — structured or unstructured. On the other hand, we want to investigate how a generic approach like RINS compares to a domain-dependent approach.

We have chosen for these aims the job-shop scheduling problem with earliness and tardiness costs. First, the problem with several resources is infamous for its difficulty and the few papers in the literature that attack this problem visibly do not solve it in an effective nor robust manner. Next, large neighborhood search has been very successfully applied to various scheduling problems [1, 17, 7, 21]. Finally, the simplest and most common MIP formulation of the problem is well-suited to RINS because MIP solvers encounter great difficulties in generating integer solutions. Nevertheless, a straight RINS application has been shown [11] to compare favorably to existing algorithms on this problem.

The remainder of the paper is organized as follows. Section 2 formally introduces the job-shop scheduling problem with earliness and tardiness costs and rapidly overviews the literature on this problem. Section 3 presents our unstructured large neighborhood search approach based on MIP and RINS. Section 4 presents our structured large neighborhood search approach embedded in constraint programming. Section 5 discusses computational results on a number of varied benchmarks. Section 6 concludes with a summary of what can be learned from this comparison.

2 The job-shop problem with earliness and tardiness costs

An $n \times m$ job-shop problem consists of a set J of n jobs and a set R of m resources of capacity one. Each job $j \in J$ consists of a set of m activities ordered according to a given permutation $\sigma_j = (\sigma_j^1, \sigma_j^2, \ldots, \sigma_j^m)$ of the resources: Job j must be executed first on resource σ_j^1, then on σ_j^2 and so on. The capacity required for each activity is one. Let p_{ji} be the processing time of job j on resource i. Each job cannot start before its release date r_j.

An admissible solution to a job-shop problem is an assignment of start times to each activity such that no pre-emption is allowed, that is once an activity has started execution, it must be executed for its entire duration without interruption; for each resource, the resource capacity constraints are not violated: each resource can execute at most one activity at a time; for each job, the first activity starts after the job release date and the activities are processed in the specified order.

Various objective functions are considered in the literature. The most studied objective is to minimize the makespan, that is minimize the completion time of the job that finishes last. In this paper, we concentrate instead on minimizing the total earliness/tardiness cost. One example of where this objective might arise is in "just-in-time" inventory management. A late job has negative consequences on customer satisfaction and time to market, while an early job increases storage costs. In fact, the earliness/tardiness problem is a particular case of the minimization of the tardiness and inventory costs, where inventory costs are supposed to hold only on the completed product. More formally, each job j is additionally characterized by its due date d_j (we suppose $d_j \geq r_j$) and two cost factors, earliness α_j if the last activity of job j ends before its due date, and tardiness β_j if it ends after its due date. Let C_j be the completion date of the last activity of job j. The cost incurred by each job j is:

$$z_j = \begin{cases} \alpha_j(d_j - C_j) & \text{if } C_j \leq d_j \\ \beta_j(C_j - d_j) & \text{if } C_j > d_j \end{cases}$$

The objective function for the entire problem is the sum of the costs incurred for each job: $\min \sum_{j \in J} z_j$.

This variant of the job-shop scheduling problem is known to be NP-hard [15].

For the single machine problem, several custom approaches exist, for example branch-and-bound [16] and dynamic programming [27].

The multiple resources problem has been much less studied. Constraint programming approaches have been applied in a very heuristic manner to problems including both tardiness and inventory costs [14, 24]. More recently, two hybrid approaches have been introduced by Beck and Refalo. HLS [4] is a combination of tabu search and linear programming: tabu search is used to choose a sequence of activities for each machine, and linear programming is used to assign optimal start times to each activity respecting the specified sequencing. It should be noted that HLS produces neither lower bounds nor optimality proofs. CRS [3, 5] is a hybrid of contraint programming and linear programming that relies on probe backtrack search [25]. It proceeds by first solving the *cost relevant subproblem* (CRS) that includes only the activities involved in the cost function (*i.e.*, the last activity of each job), next trying to extend the CRS solution to a solution to the whole problem, and finally branching as in probe backtrack search when no CRS extension is found. CRS provides lower bounds and optimality proofs. Last, three recent MIP strategies have been applied to a straightforward modelization of the multiple resources problem [11].

As emphasized in [11], CRS is particularly effective on instances where the CRS can be solved and extended rapidly. On other instances, it is significantly outperformed by HLS and MIP approaches.

3 Unstructured LNS in Mixed Integer Programming

This section presents our unstructured LNS approach. We first describe relaxation induced neighborhood search. We then present the basic MIP model we used and how we improved it.

3.1 Relaxation Induced Neighborhood Search (RINS)

Relaxation Induced Neighborhood Search (RINS) is a generic mixed integer programming (MIP) heuristic first introduced in [10]. It is a large neighborhood search that uses the continuous relaxation to define its neighborhood. RINS can be used on any MIP model without any other input than the model itself, it does not need any information on the high level structure of the problem at hand. To our knowledge, it is therefore the first *unstructured* LNS — besides random choice, which is often not effective. Note however that RINS is not completely unstructured, it does not use explicitly the high level structure of the problem but the continuous relaxation helps to capture it.

The idea behind RINS is that, at any node of a branch-and-cut tree, the best integer solution found so far (the *incumbent*) and the continuous relaxation each achieve one and fail to achieve one of the following conflictual objectives: integrality and optimality. RINS is based on the intuition that decisions (*i.e.*, instantations of variables) common to the incumbent and the continuous relaxation form a partial solution that is likely to be extended towards a complete solution that achieves integrality and reaches or comes near to optimality. Therefore, it focuses attention on those variables that differ in the continuous relaxation and in the incumbent, which are intuitively the ones that appear to merit further attention. Our RINS algorithm is thus simple. At a node of the global branch-and-cut tree, the following steps are performed:

1. Fix the variables that have the same value in the incumbent and in the current continuous relaxation;
2. Solve a sub-MIP on the remaining variables.

The sub-MIP is potentially large and itself difficult to solve. Therefore, its exploration has to be truncated in order to control the time spent in RINS. We do so by setting a node limit nl, for example $nl = 1000$. While RINS can be called at each node of the global branch-and-cut tree, providing automatic neighborhood diversification, the differences between consecutive nodes in the tree are typically quite small, leading to very similar neighborhoods. We have found that it is preferable to apply it only every f nodes, for example every 100 nodes.

One major strength of RINS is that it explores a neighborhood both of the incumbent and of the continuous relaxation. Without outside information, it can never be known with certainty which of the two is closer to the optimal integer solution. In RINS, the incumbent and the continuous relaxation play perfectly symmetrical roles, so if one is of poor quality, the other will automatically help to define a fruitful neighborhood and vice versa.

For the experiments described in Section 5, we also used a tree traversal strategy called *guided dives* which was also introduced in [10]. Its principle is to explore nodes in an order that guides the search towards the neighborhood of the incumbent, a region of the search space that is expected to contain good integer solutions. It has been shown on a variety of MIP models [10, 11] that RINS alone almost consistently performs better than guided dives alone. However, we found out that combining RINS and guided dives, that is using RINS plus guided dives as the tree traversal strategy for the global MIP, allows us to increase robustness throughout the 1.0-looseness random problems set described in Section 5.1. It did not significantly improve performance on the problems from the genetic algorithm literature described in 5.2.

3.2 The disjunctive MIP formulation of Applegate and Cook [1]

Various MIP formulations exist for the job-shop (see [5] for a quick survey). We chose the disjunctive formulation of Applegate and Cook [1] because it is the simplest and most common formulation. It is also the formulation we used in [11].

Let continuous variable x_{ji} represent the start time of the processing of job j on resource i. Minimizing the sum of earliness and tardiness costs is modeled by $\min \sum_{j \in J} z_j$ where variable z_j represents the cost incurred by job j:

$$\begin{cases} z_j \geq \alpha_j (d_j - x_{j\sigma_j^m} - p_{j\sigma_j^m}) \\ z_j \geq \beta_j (x_{j\sigma_j^m} + p_{j\sigma_j^m} - d_j) \end{cases} \quad \forall j \in J$$

The release dates constraints are simply:

$$x_{j\sigma_j^1} \geq r_j, \ \forall j \in J \tag{1}$$

The precedence constraints between the activities in each job are modeled by the constraints:

$$x_{j\sigma_j^{t+1}} \geq x_{j\sigma_j^t} + p_{j\sigma_j^t}, \ \forall j \in J, \ \forall t = 1 \dots m-1 \tag{2}$$

The resource constraints are modeled by the constraints: $\forall\ p < q \in J, \forall\ i = 1 \dots m$,

$$\begin{aligned} x_{pi} &\geq x_{qi} + p_{qi} - M y_{pq}^i && (3)\\ x_{qi} &\geq x_{pi} + p_{pi} - M(1 - y_{pq}^i) && (4)\\ y_{pq}^i &\in \{0,1\} \end{aligned}$$

where M is some large constant and the y_{pq}^i variables are each constrained to take a value of either 0 or 1. The interpretation is that $y_{pq}^i = 1$ if job p is scheduled before job q on resource i and $y_{pq}^i = 0$ if p is scheduled after job q on resource i. This type of model is known as a *big-M formulation*. It is expected to behave poorly, *i.e.*, usually to have a loose continuous relaxation or as shown in [11] to make it difficult for a MIP solver to find good integer solutions because it very weakly links the decision variables y that appear in the resource constraints and the secondary variables x that appear in the precedence constraints.

3.3 Tightening the big-M formulation

Even with this straightforward big-M MIP model, RINS and guided dives compare favorably [11] to other approaches such as CRS and HLS. Nevertheless, substantial improvements in the model are possible. One way to improve this model is to add cuts, *i.e.*, valid inequalities that cut off the continuous relaxation solution while retaining all integer feasible solutions. For example, Applegate and Cook [1] define a number of cutting planes, which allows the continuous relaxation to become a better approximation of the convex hull of the integer solutions.

In this paper, we chose another way to tighten this model by concentrating on how to choose better big-M's constants. M needs to be a majorant of $x_{qi} + p_{qi} - x_{pi}$, for each pair (p, q) of different jobs and for each resource i. In other words, M needs to be a majorant of the difference between the end time and the start time of each pair of activities. Otherwise, M should be chosen as small as possible in order to tighten the link between the x and y variables.

First we can choose as in [11]

$$M = \max_{j \in J} d_j + \sum_{j \in J} \sum_{i=1}^{m} p_{ji} \tag{5}$$

There exists indeed a schedule that finishes before this value and any schedule that finishes after this value would bear a greater earliness/tardiness cost.

Note that M need not take the same value for each resource constraint. Let M_{pq}^i and M_{qp}^i be the big-M constants for resource constraint defined by Equation 3 and 4 respectively. M_{pq}^i need now only be a majorant of $x_{qi} + p_{qi} - x_{pi}$. This remark allows us to compute better M values:

$$M_{pq}^i = \max_{j \in J} d_j + \sum_{j \in J} \sum_{i=1}^{m} p_{ji} - E_{pi} - F_{qi} \tag{6}$$

where E_{ji} denotes the earliest possible start time of job j on resource i (j's release date plus the sum of the processing times of j on the resources ranked before i in j prescribed ordering) and F_{ji} denotes the time for completion of job j after it has been processed on resource i (the sum of j's processing times on the remaining resources after i).

The model can be further tightened when an integer solution is known. M^i_{pq} need then only be a majorant of the possible values of $x_{qi}+p_{qi}-x_{pi}$ in the schedules of total earliness/tardiness cost smaller than the cost z^{inc} of the incumbent. In any improved solution, the tardiness cost of job q is smaller than z^{inc}, therefore $C_q \leq d_q + \frac{z^{inc}}{\beta_q}$. Besides, by definition of F, $x_{qi} + p_{qi} + F_{qi} \leq C_q$. By definition of E, we also have $x_{pi} \geq E_{pi}$. Finally, we obtain

$$x_{qi} + p_{qi} - x_{pi} \leq d_q + \frac{z^{inc}}{\beta_q} - F_{qi} - E_{pi} \tag{7}$$

which defines the new value of M^i_{pq}.

Another way of computing new values for M's is the following. In any improved solution, the sum of the earliness cost for job q and of the tardiness cost for job p is smaller than z^{inc}: $\beta_q(C_q - d_q)^+ + \alpha_p(d_p - C_p)^+ \leq z_{inc}$. Let $\gamma_{pq} = \min(\beta_q, \alpha_p)$. The previous equation implies that $\gamma_{pq}(C_q - d_q + d_p - C_p) \leq z^{inc}$. If i is the last resource on which p is to be executed, $x_{pi} + p_{pi} = C_p$. Using the definition of F_{qi}, we obtain:

$$x_{qi} + p_{qi} - x_{pi} \leq d_q + \frac{z^{inc}}{\gamma_{pq}} - F_{qi} - d_p + p_{pi} \tag{8}$$

which defines the new value of M^i_{pq} in this particular case.

The new M values computed thanks to the incumbent can be used either to add cuts corresponding to the updated resource constraints, or to build a new MIP model and restart optimization from scratch, keeping only the incumbent from the previous iteration. The first option has the advantage of not losing information contained in the branch-and-cut tree, whereas the second option might explore the same nodes several times. However, if the updated M values computed thanks to the new integer solution are dramatically better — which is often the case in practice, at least between the first and second iteration — the tree traversal strategy in the new MIP is likely to be significantly better than that of the previous MIP because it corresponds to a much tighter MIP model. Furthermore, the first option drawback is that it adds at each step a number of additional constraints, which slows down the resolution of the linear program at each node of the branch-and-cut tree. Our algorithm thus proceeds by the following steps:

1. Start with a MIP model with the M's as defined by Equation 6.
2. Start from scratch, reusing only the incumbent from the previous iteration (if it exists), and solve the current MIP model by branch-and-cut and relaxation induced neighborhood search, with guided dives as global tree traversal strategy as defined in Section 3.1. Stop if a proof of optimality has been obtained, or if more than 1000 nodes have been explored and if the value of the upper bound has decreased by more than 20%. This second stopping criterion allows us to change models only if the incumbent has improved enough to let us expect the new M values to be worth discarding the history of the branch-and-cuts nodes explored.
3. Discard the current MIP model and create a new MIP model with new M values computed according to Equations 7 and 8. Go to step 2.

4 Structured LNS in Constraint Programming

This section presents our structured LNS implementation. We describe first the neighborhoods used to perform large neighborhood search. This allows us to rank activities on each resource. We present next the linear program that assigns a start time to each activity at every LNS iteration when a sequence of activities on each resource has been obtained. We describe last our adaptive iterative solving architecture.

4.1 Specialized neighborhoods for scheduling

Each neighborhood corresponds to a subset of activities that are released. The order of activities in this subset can be changed while the sequence of the rest of the activities is fixed as in the current solution. The sub-problem corresponding to each neighborhood is solved by a ranking algorithm implemented in ILOG Scheduler 5.3.

We distinguish between two types of neighborhoods. The first type (*loosely structured neighborhoods*) only relies on the job-shop structure of the problem, while the second type (*cost structured neighborhoods*) exploits the specific structure of the earliness/tardiness objective function.

In the following, let s be a parameter that roughly indicates the size of the neighborhood.

4.1.1 Loosely structured neighborhoods.

We have implemented four loosely structured neighborhoods.
Random: This neighborhood is completely unstructured and generic. It releases s randomly chosen activities.
Resource based: This neighborhood releases all activities on given resources. If l is the average number of activities on each resource, this neighborhood releases s/l resources. It is inspired by the shifting bottleneck procedure [17].
Random Time Window: This neighborhood releases activities scheduled within different time windows on different resources. More precisely, until s activites have been released, this neighborhood selects a resource that has not been selected before, it chooses two activities on this resource and it releases all activities that are scheduled in between on this resource. It should be noted that there is no correlation between the time windows selected for different resources.
Consecutive Pair: This neighborhood releases $s/2$ pairs of consecutive activities, *i.e.* that are scheduled on the same resource one after the other in the current solution.
These are very simple neighborhoods. Note that they can be applied directly to any scheduling problem with unary resources and extended easily to resources with capacity greater than one. Note also that all these neighborhoods rely heavily on randomness in their definition.

4.1.2 Cost structured neighborhood.

This neighborhood first selects two jobs p and q that are not scheduled on time in the current solution: the costs z_p and z_q incurred by each job are non-zero. This neighborhood releases each activity of job p and q, plus some other activities scheduled after or before each in the current solution, depending on whether the corresponding job is early or late. The aim is to allow the corresponding job to be pushed left or right in order to decrease its tardiness or its earliness cost.

More formally, for each activity a of job p and q, let $rank(a)$ be its rank in the current sequence of activities on the corresponding resource. We additionally release all activities b that execute on this same resource and whose rank verify:

$$rank(a) - r + \delta \leq rank(b) \leq rank(a) + r + \delta$$

where for each $j \in \{p, q\}$, δ_j is equal to 1 if job j is early and -1 if it is late. r can be interpreted as a radius of the neighborhood. We have arbitrarily chosen $r = s/15$.

4.1.3 Why loosely structured neighborhoods?

Experiments showed that removing any of the five neighborhoods described above causes a degradation in performance. It is especially interesting to note that the results are much worse if only the cost-structured neighborhood is used. In that sense, our set of neighborhoods is minimal. Our explanation is that the cost structured neighborhood does not provide enough diversification for the LNS approach. Only the combined strength of the cost structured neighborhood and of the loosely structured neighborhoods offers enough diversification to escape local minima. This is consistent with results found in [22] where randomness is needed at the heart of the LNS strategy to provide robustness.

4.2 The LP-Based Time Placement Algorithm

Each iteration of LNS provides a sequence that states in which order activities are to be executed on each resource. To build a complete schedule, one must additionally assign a start time to each activity. This is a crucial element of our strategy because it is not optimal to flush each activity towards the start of the schedule as when minimizing the makespan. Time placement is computed by solving a linear program each time all activities have been ranked. The model is a simplified version of the MIP model described in Section 3.2 as all disjunctions have been resolved by the neighborhood search.

As in Section 3.2, let continuous variable x_{ji} represent the start time of the processing of job j on resource i. The release dates constraints and the precedence constraints are modeled as in Section 3.2 by Equation 1 and 2. Minimizing the sum of earliness and tardiness costs is modeled as in Section 3.2 by $\min \sum_{j \in J} z_j$ where variable z_j represents the cost incurred by job j:

$$z_j \geq \alpha_j (d_j - x_{j\sigma_j^m} - p_{j\sigma_j^m}) \tag{9}$$

$$z_j \geq \beta_j (x_{j\sigma_j^m} + p_{j\sigma_j^m} - d_j) \tag{10}$$

The difference with the model of Section 3.2 is that the resource constraints are now modeled by the constraints: $\forall\ i = 1 \dots m, \forall\ p, q \in J$ such that job p is ranked after job q on resource i according the sequence obtained by LNS,

$$x_{pi} \geq x_{qi} + p_{qi}$$

This allows us to eliminate all integer variables. We now simply have a linear program to solve.

If $\alpha_j = \beta_j\ \forall j \in J$, the objective function can be rewritten as $\min \sum_{j \in J} \alpha_j t_j$, where t_j is the amount of time by which job j is early or late:

$$t_j \geq (d_j - x_{j\sigma_j^m} - p_{j\sigma_j^m}) \tag{11}$$

$$t_j \geq (x_{j\sigma_j^m} + p_{j\sigma_j^m} - d_j) \tag{12}$$

Constraints 11 and 12 replace Constraints 9 and 10. In that way, the constraints matrix becomes totally unimodular. Therefore, all solutions are integer.

In the general case, α_j and β_j are not necessarily equal. However, each extreme point of the polyhedron corresponds to an assignment in which each job ends before or after its due date. It is therefore also an extreme point of the polyhedron defined by the matrix where only one constraint among Constraints 9 and 10 is active for each job $j \in J$. This matrix can be transformed into a totally unimodular matrix as in the particular case detailed above. Therefore, each extreme point corresponds to an integer solution. Hence solving only this linear program yields an integer solution.

4.3 The large neighborhood search architecture

As presented in the introduction, choosing the size of the neighborhood (here the parameter s) is a challenge in itself. To keep the algorithm simple and yet robust, we start with $s = 50$ and increase s when repeated failures happen. More precisely, after 50 consecutive failures, s is increased by 1 and the counter of successive failures is reset to 0. Note that other arbitrary choices of the number of maximum consecutive failures also give good results. s is furthermore constrained to never exceed 150, 3 times the initial size of the neighborhoods.

Finally, our architecture benefits from previous work on fast restart policy [23]. It demonstrated that the standard fast restart policy used at each iteration to control the time spent in the sub-problem could be improved. In practice, the default Depth-First Search with a fail limit was replaced by a combination of DBDFS(1) [2] with a fail limit that grows slowly after each unsuccessful iteration.

5 Computational results

We now present experimental results. The unstructured LNS approach **uLNS** presented in Section 3.1 uses a modified version of CPLEX 8.1. The structured LNS approach **sLNS** presented in Section 4 uses ILOG Solver 5.3 and ILOG Scheduler 5.3 and ILOG CPLEX 8.1. All experiments were done on a 1.5GHz Pentium IV system. More precisely, the algorithms compared are:
uLNS v1: The first unstructured LNS implementation described in Section 3.1, using only RINS and M given by Equation 5.
uLNS v2: The complete unstructured LNS implementation.
sLNS v1: The first naive implementation of structured LNS using only the pure random neighborhood.
sLNS v2: The implementation of structured LNS with the LP-based time placement algorithm but without the cost structured neighborhood.
sLNS v3: The implementation of structured LNS using all five neighborhoods, but without the LP-based time placement algorithm.
sLNS v4: The complete implementation of structured LNS.

Because our experiments generated a great volume of data, and because a number of problems we studied are still open, we often aggregate results by giving for each algorithm the geometric mean of the ratio (abbreviated GMR) obtained by dividing the cost of the solution obtained with this algorithm by the best known upper bound.

5.1 Random problems

This benchmark [3] consists of 90 randomly generated job-shop earliness/tardiness scheduling problems of size 10×10, 15×10 and 20×10, divided in three sets depending on the value of parameter called the *looseness* (see [11] for a discussion on the influence of this parameter).

For all problems with looseness 1.3 and 1.5, **sLNS v4** and **uLNS v2** find (and prove for the latter) the optimal solutions very easily. We concentrate therefore on the more difficult 30 1.0-looseness problems.

For the whole set and for each subset of 10 problems with 10, 15 and 20 jobs, for each of our algorithms, we now report the sum of the cost of the solutions obtained, the GMR and the number of optimality proofs. Note that our unstructured LNS approach embbeded in constraint programming is an incomplete method, it neither provides lower bounds nor optimality proofs. The *best* column gives the best results found either by one of our algorithms or by CRS-All [3]. The utmost right column *LB* gives the lower bound obtained by our MIP-based approaches. All results were obtained in 20 minutes on our test system, except for **uLNS v2-2h** the complete unstructured LNS algorithm that was run for 2 hours.

	uLNSv1	uLNSv2	uLNSv2-2h	sLNSv1	sLNSv2	sLNSv3	sLNSv4	best	LB
GMR all	11.11	3.35	2.2	3.07	2.33	1.67	1.43		
SUM all	45,1476	156,001	97,804	122,890	89,577	62,795	52,307	36,459	11,407
#PROOFS all	0	0	2	-	-	-	-		
GMR 10	2.75	1.21	1.09	1.62	1.36	1.23	1.06		
SUM 10	51,191	24,233	20,919	31,875	27,644	24,086	20,323	18,936	3,070
#PROOFS 10	0	0	0	-	-	-	-		
GMR 15	14.63	3.08	2.27	3.58	2.21	1.75	1.49		
SUM 15	103,635	24,270	16,464	47,443	20,304	12,423	16,390	7,154	4,333
#PROOFS 15	0	0	1	-	-	-	-		
GMR 20	34.1	10.11	4.3	3.58	4.23	2.16	1.85		
SUM 20	296,650	107,498	60,421	43,572	41,629	22,319	19,561	10,369	4,004
#PROOFS 20	0	0	1	-	-	-	-		
BEST	0	4	9	2	7	8	14		

Let us recall the results obtained on the same test system with the same time limit by other algorithms: default CPLEX on the basic model of **uLNS v1**, a simple constraint programming

approach (default CP) [3], CRS-All [3] and HLS [4][1].

	defaut CPLEX	default CP	CRS-All	HLS
GMR all	18.59	26.56	10.60	13.98
SUM all	654,290	1,060,634	885,546	478,181

We can draw a few basic conclusions from this experimental data. First, both our LNS approaches outperform significantly all previous algorithms. Note that the earliness/tardiness objective function has no fixed cost, it can be very small or even reach 0 on some problems[2] and, in turn, each wrong decision can shift a number of jobs away from their due dates, hence having a great impact on the cost function. This explains in part the huge differences in performance among algorithms.

The structured LNS approach, even in its most simple form **sLNS v1**, improves dramatically on default CP performance. This can be explained by the fact that constraint propagation is weak on a sum objective like the total earliness/tardiness cost, in contrast to a max objective like the makespan, because it cannot relate the sum cost to one particular variable to work on. LNS allows us to isolate a few terms of the sum, therefore a few variables to concentrate on, whose effects on the cost function are much more direct to evaluate. This is further reinforced by the cost structured neighborhood used in **sLNS v3** and **sLNS v4**. Constraint propagation is also stronger in LNS sub-problems than in the global problem because the sub-problems are naturally smaller, and more constrained because fixing activities brings additional constraints.

The unstructured LNS approach based on RINS is outperformed on this benchmark by all structured LNS approaches. It should however be noted that RINS is competitive on the smallest problems with 10 jobs and that performance degrades rapidly as the size of the instances increases. Our proposed explanation is that this approach carries out many fewer LNS iterations because it also solves linear programs at each node of the MIP tree, which consumes time, especially on the bigger instances. This explanation is confirmed by the improvement obtained when giving **uLNS v2** a 2-hour time limit, which is especially visible on the 20-job instances.

This last remark mitigates the embarrassing superiority of the random neighborhoods of **sLNS v1** on the more sophisticated RINS approach. The good results of **sLNS v1** can be explained in three ways. First, it is not a naive random approach because it benefits from the powerful and specific ILOG Scheduler algorithms to solve each LNS sub-problem. Note that the good results obtained by all structured LNS versions when compared to RINS may be explained either by a better definition of the neighborhoods or by a better resolution of the LNS sub-problems (ILOG Scheduler *vs.* default CPLEX on a basic MIP model). Next, this shows the strength of the LNS paradigm. Indeed, relying on simple randomization to define neighborhoods succeeds because the large size of the neighborhoods provides diversification and the powerful algorithm used to solve sub-problems provides intensification. Finally, many instances of this randomly generated benchmark appears to be quite unconstrained, with much idle time on resources and activities quite free to be shifted or inverted. This is the most favorable case for randomly chosen neighborhoods. In contrast, the unstructured LNS approaches are deterministic. This is very likely to hinder their diversification abilities.

It should be finally noted that the best algorithm **sLNS v4** is on average still about 40% above the best known solutions, which indicates on the one hand that no algorithm is yet robust over the whole benchmark, and on the other hand that this randomly generated benchmark probably exhibits a number of numerical singularities. The very small number of optimality proofs (7 out of 30) and the important gap between the best known lower bounds and the best known upper bounds further indicates that no algorithm is yet extremely effective.

[1] HLS results were obtained on a 4 to 5 times slower computer.

[2] This does not happen on any of the instances studied in this paper.

5.2 Problems from the genetic algorithms literature

5.2.1 Original benchmark

These problems are provided by Morton and Pentico [19] and have been used in a number of studies of GA scheduling [28]. We compare our algorithms to the best results obtained by various genetic algorithms as reported in [28]. All our results were obtained in two hours on our test system. For each algorithm, we provide the upper bound obtained and GMR over the whole set (GMR 12) and over the five open problems (GMR 5). A * mark indicates that the algorithm proves optimality in the allowed time.

Problem	size	uLNSv1-UB	uLNSv2-UB	uLNSv2-LB	sLNSv1	sLNSv2	sLNSv4	GA-best
jb1	10x3	0.191	0.191*	0.191*	0.191	0.191	0.191	0.474
jb2	10x3	0.137	0.137*	0.137*	0.174	0.138	0.137	0.499
jb4	10x5	0.568	0.568*	0.568*	0.568	0.568	0.568	0.619
jb9	15x3	0.333	0.333*	0.333*	0.639	0.334	0.333	0.369
jb11	15x5	0.213	0.213*	0.213*	0.215	0.215	0.213	0.262
jb12	15x5	0.190	0.190*	0.190*	0.191	0.191	0.190	0.246
ljb1	30x3	0.215	0.215*	0.215*	0.216	0.216	0.215	0.279
ljb2	30x3	0.508	0.508	0.230	0.613	0.508	0.508	0.598
ljb7	50x5	0.143	0.123	0.058	0.128	0.141	0.110	0.246
ljb9	50x5	1.168	1.270	0.124	0.919	0.915	1.015	0.739
ljb10	50x8	0.621	0.558	0.197	0.638	0.655	0.525	0.512
ljb12	50x8	0.682	0.488	0.053	0.583	0.510	0.605	0.399
GMR 12		1.13	1.08	0.60	1.19	1.09	1.07	1.41
GMR 5		1.34	1.21	0.29	1.26	1.21	1.16	1.21

The most striking fact on this benchmark is that both our LNS approaches outperform significantly genetic algorithms. The 3 biggest instances `ljb9`, `ljb10`, and `ljb12`, show that genetic algorithms are able to decrease quickly the objective cost because they can run a huge number of cheap and very diversified iterations. In contrast, our LNS approaches suffer from the cost of calling ILOG Scheduler or solving a sub-MIP at each LNS iteration. However, on the smaller `jb*` and `ljb1` problems, genetic algorithms are far from reaching optimality whereas structured LNS reaches and MIP proves optimality in a few seconds. Performing a great number of simple moves is indeed not enough to reach optimality; one needs to perform more intelligent moves to eliminate the last difficulties of each problem. The risk of relying on random moves only is further illustrated by the extremely poor performance of the random structured LNS **sLNS v1** on one instance (`jb9`).

As in the previous benchmark, structured LNS outperforms unstructured LNS but the difference in performance is much less apparent.

5.2.2 Variations

The last `ljb` problems with 50 jobs appeared to be too difficult for a MIP solver to come near an optimality proof. Therefore we extracted from `ljb7`, `ljb9`, `ljb10`, and `ljb12` smaller problems with all resources but only 30 jobs each that are chosen in 7 different ways as detailed in the following. This allows us to generate 28 new problems with various characteristics, whose size is closer to the size of the random benchmark instances, and for which the lower bounds obtained with the MIP approach are more meaningful than on the original problems. The instances are available on request from the authors. For Series 1 to 6, the jobs are first sorted according to their due dates.

Serie 1: Keep only the first 30 jobs.
Serie 2: Keep only the last 30 jobs.
Serie 3: Skip regularly jobs to keep in the end only 30 jobs.
Serie 4: Select 30 jobs in 5 batches of 6 jobs that have close due dates.
Serie 5: Select 30 jobs in batches that have close due dates as in **Serie 4**. In each batch, replace the due dates by the average due date in the batch.

Serie 6: Delete some jobs so that the sequence of the resources for each remaining job is very similar (ideally a perfect flowshop, else with the same first and sometimes second machine)
Serie 7 contains all 50 jobs but the release dates are removed (each job can start at 0).

All the following results are again obtained in 2 hours on our test system. We give the number of optimality proofs produced by **uLNS v2** and the GMR, aggregating the results first by original problem from which the new instances are extracted, then by series. The best upper bounds used to compute the GMR were obtained by one of our algorithms.

Problem Set	uLNSv2-UB	uLNSv2-LB	#PROOFS	sLNSv4
ljb7	1.01	0.84	3	1.07
ljb9	1.07	0.54	2	1.04
ljb10	1.01	0.79	2	1.02
ljb12	1.08	0.29	2	1.10
Serie-1	1.00	0.57	0	1.08
Serie-2	1.01	0.44	0	1.18
Serie-3	1.00	1.00	4	1.01
Serie-4	1.00	1.00	4	1.05
Serie-5	1.00	0.74	0	1.02
Serie-6	1.01	0.72	1	1.05
Serie-7	1.30	0.14	0	1.00

As expected, Series 3 and 4 are the easiest. RINS performs better than structured LNS on all series, except Serie 7. This is easily explained by Equation 7 used in the general case to tighten the MIP model. Indeed, setting all release dates to 0 decreases significantly the earliest start time E_{ji} for each job j on resource i, hence the MIP model cannot be tightened much.

5.3 One-machine problems

One-machine problems are the most degenerate case of job-shop scheduling problems, and also the most studied particular case of scheduling problems with earliness and tardiness costs. Our aim is to compare in this section a dynamic programming algorithm (DP) [27] that is specific to the one-machine problem with our more generic algorithms. The benchmark used in this section is under construction. It will gather instances reproducing the structure of actual industrial problems (submitted by ILOG customers), and a few randomly generated instances or well-known instances from the literature. Note that some of these problems have many more activities than the instances in the previous benchmarks. However, a number of activities are in fact identical (same processing time, release date and due date) and there exists only a small number of different families of identical activities. All algorithms presented here exploit this feature by breaking symetries (for example adding precedence constraints between the activities of the same family). All the following results are obtained in 2 hours on our test system.

Problem	size	best-UB	best-LB	DP-UB	DP-LB	uLNSv2-UB	uLNSv2-LB	sLNSv1	sLNSv4
NCOS-01	8	1025	1025	1025	1025	1025	1025	1025	1025
NCOS-04	10	2504	2504	2504	2504	2504	2504	2740	2504
NCOS-05	15	4491	4491	4491	4491	4491	3264	6063	4491
NCOS-11	20	8077	8077	8077	8077	8077	8077	11054	9160
NCOS-12	24	10876	10847	10876	10847	10904	3084.23	11217	10889
NCOS-13	24	6878	6878	6878	6878	6893	3807.99	7361	6895
NCOS-15	30	16579	16579	16579	16579	16579	4265	29381	18464
NCOS-31a	75	28250	9104.57			28250	9104.57	47630	29675
NCOS-32	75	35370	35086		35086	35370	18213	35570	35370
NCOS-51	200	62567.8	61745.8		61745.8	62699	39266.4	64373.4	62567
NCOS-61	500	1732636	1730110		1730110	1732636	1675580	1767300	1767290
GMR			0.90			1.00	0.59	1.19	1.03

The main conclusion on this benchmark is that our LNS approaches are competitive with the dynamic programming algorithm, for generating upper bounds as well as lower bounds (for the

unstructured LNS only). They are more robust than dynamic programming when the size for the problem increases.

On this benchmark, RINS outperforms both versions of structured LNS. The structured LNS approach **sLNS v1** that relies only on random neighborhoods seems to reach its limits on this benchmark, probably because this problem is more degenerate, and hence needs more carefully chosen neighborhoods.

5.4 Minimizing maximum lateness

Our last benchmark comes from Demirkol, Mehta, and Uzsoy [12, 13]. The main difference between this benchmark and the benchmarks studied in the previous sections is the objective function. Instead of a weighted sum of earliness/tardiness costs, the function to minimize here is the maximum lateness over all jobs: $\min \max_{j \in J} C_j - d_j$. We have chosen to concentrate on the 20×15 instances which are comparable in size to the instances studied in the rest of the article. This represents a total of 60 instances divided in three sets of 20 instances, each set corresponding to a different structure: job-shop, job-shop with a workflow structure (the resources are divided in two sets, each job has to be processed first on each resource of the first set, then on all resources of the second set), and flow-shop.

The following results were obtained with a one-hour time limit on our test system for each instance. We also report the results provided by [12, 13] in the "Orig" column.

	Average			GMR		
	sLNSv4	uLNSv2	Orig	sLNSv4	uLNSv2	Orig
all	2,176.87	2,455.53	2,349.78	1.00	1.12	1.08
jobshop	1,711.00	1,836.50	1,912.65	1.00	1.08	1.12
2-set jobshop	2,261.90	2,612.75	2,427.70	1.00	1.15	1.08
flowshop	2,557.70	2,917.35	2,709.00	1.00	1.14	1.06

Results from [12, 13] do not use a comparable time limit, nevertheless both our LNS approaches provide competitive results, showing that they are robust with respect to this new objective function. This is especially true for structured LNS. Our first explanation for the superiority of structured LNS was that a "max" objective propagates more in constraint programming and is less effective in mixed integer programming. But the same phenomenon is witnessed when minimizing the sum of lateness: $\min \sum_{j \in J} (C_j - d_j)$, as shown in the following table.

	Average		GMR	
	sLNSv4	uLNSv2	sLNSv4	uLNSv2
all	34,892.70	39,107.25	1.00	1.12
jobshop	28,550.90	31,696.75	1.00	1.11
2-set jobshop	38,509.90	44,177.55	1.00	1.15
flowshop	37,617.30	41,447.45	1.00	1.10

Note that the sum of lateness obtained is very close to the number of jobs multiplied by the maximum lateness obtained in the previous set of experiments. This indicates that all jobs are about equally late, the problem is hence quite uniform. Besides, there are no release dates as on the random problems of Section 5.1 and the Series 7 of section 5.2, therefore the MIP model is more difficult to tighten. This might explain why the RINS performs worse on this benchmark.

6 Conclusion

In this paper we have shown that large neighborhood search is a powerful paradigm to solve hard combinatorial problems such as the job-shop scheduling problem with earliness and tardiness costs. Both our approaches of structured and unstructured LNS, sometimes even in their least sophisticated form, have proved to be effective and robust — outperforming a variety of existing algorithms on a number of benchmarks corresponding to different variants of the job-shop scheduling problem with earliness and tardiness costs.

Note in particular that our structured LNS embedded in constraint programming dramatically improves on pure constraint programming by defining a specific neighborhood that exploits the structure of the earliness/tardiness. This allows to attack successfully a sum objective, on which constraint propagation was traditionnally very weak.

Our structured and unstructured LNS approaches yield competitive results on every benchmark, each approach outperforming the other on some benchmarks and vice versa. We propose four elements to explain the difference of performance between the two approaches. The first is how neighborhoods are defined, using explicitly the high level structure of the problem or not. The second is how the neighborhoods are explored, using constraint programming or branch-and-cut. In order to isolate each of these possible causes, future work will be directed at implementing structured LNS in a MIP framework. The third explanation we propose is that RINS defines neighborhoods in a deterministic manner, whereas our structured LNS approach relies heavily on randomization. This allows to diversify the search in a simple yet effective way. Finally, our fourth explanation is that our unstructured LNS approach not only consists in RINS but also in exploring a global branch-and-cut tree as when solving any MIP. Therefore not the whole computation time is devoted to finding upper bounds, but a significant part of the time is spent in branching and solving continuous relaxations at each node of the global branch-and-cut tree. In turn, this allows to compute lower bounds and to produce optimality proofs, which is not possible with our structured LNS approach.

It should finally be noted that none of our approaches is totally generic. This is obvious for structured LNS: its neighborhoods depend heavily on the specific problem at hand. Though less apparent, this is also true for RINS. RINS uses not other input than the MIP model itself, and hence is generic to any MIP. But the improvement of our results with successive versions of our MIP model show that tightening the MIP model is a important element of the unstructured LNS strategy. For RINS, the problem-specific part of the work is the definition of the MIP model instead of the definition of the tailored neighborhoods as in structured LNS.

Acknowledgments

The authors wish the express their heartfelt thanks to Philippe Refalo who provided the LP-based time placement code used in the structured LNS approach. We are also indebted to Francis Sourd for his work on one-machine problems. Finally, we would like to thank Claude Le Pape for his support and his deep thinking that helped us with this work.

References

[1] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

[2] J. C. Beck and L. Perron. Discrepancy-Bounded Depth First Search. In *Proceedings of CP-AI-OR 00*, March 2000.

[3] J. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. In *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'01)*, pages 175–188, 2001.

[4] J. C. Beck and P. Refalo. Combining local search and linear programming to solve earliness/tardiness scheduling problems. In *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 221–235, 2002.

[5] J. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research*, 118(1-4):49–71, 2003.

[6] R. Bent and P. V. Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. Technical Report CS-01-06, Brown University, september 2001.

[7] Y. Caseau and F. Laburthe. Effective forget-and-extend heuristics for scheduling problems. In *Proceedings of the First International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'99)*, 1999.

[8] A. Chabrier, E. Danna, and C. Le Pape. Coopération entre génération de colonnes avec tournées sans cycle et recherche locale appliquée au routage de véhicules (in french). In *Huitièmes Journées Nationales sur la résolution de Problèmes NP-Complets (JNPC'2002)*, 2002.

[9] A. Chabrier, E. Danna, C. Le Pape, and L. Perron. Solving a network design problem. *To appear in Annals of Operations Research, Special Issue following CP-AI-OR'2002*, 2003.

[10] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. Technical report, ILOG, 2003.

[11] E. Danna, E. Rothberg, and C. Le Pape. Integrating mixed integer programming and local search: A case study on job-shop scheduling problems. In *Proceedings of the Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'03)*, 2003.

[12] E. Demirkol, S. Mehta, and R. Uzsoy. A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics*, 3:111–137, 1997.

[13] E. Demirkol, S. Mehta, and R. Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109:137–141, 1998.

[14] M. S. Fox and S. F. Smith. ISIS: A knowledge-based system for factory scheduling. *Expert Systems*, 1(1):25–49, 1984.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[16] J. Hoogeveen and S. van de Velde. A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time. *INFORMS Journal on Computing*, 8:402–412, 1996.

[17] E. B. J. Adams and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[18] C. Le Pape, L. Perron, J.-C. Régin, and P. Shaw. Robust and parallel solving of a network design problem. In P. V. Hentenryck, editor, *Proceedings of CP 2002*, pages 633–648, Ithaca, NY, USA, September 2002.

[19] T. Morton and D. Pentico. *Heuristic Scheduling Systems*. John Wiley and Sons, 1993.

[20] M. Palpant, C. Artigues, and P. Michelon. A heuristic for solving the frequency assignment problem. In *XI Latin-Iberian American Congress of Operations Research (CLAIO)*, 2002.

[21] M. Palpant, C. Artigues, and P. Michelon. Solving the resource-constrained project scheduling problem by integrating exact resolution and local search. In *8th International Workshop on Project Management and Scheduling PMS 2002*, pages 289–292, 2002.

[22] L. Perron. Parallel and random solving of a network design problem. In *Workshop on Random Search of AAAI Conference*, 2002.

[23] L. Perron. Fast restart policies and large neighborhood search. In *Proceedings of CPAIOR 2003*, 2003.

[24] N. Sadeh and M. S. Fox. Variable and value ordering heuristics for activity-based job-shop scheduling. In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, 1990.

[25] H. E. Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

[26] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Proceeding of CP '98*, pages 417–431. Springer-Verlag, 1998.

[27] F. Sourd. Scheduling a sequence of tasks with general completion costs. Technical Report 2002/013, LIP6, 2002.

[28] M. Vasquez and L. Whitley. A comparison of genetic algorithms for the dynamic job shop scheduling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 1011–1018, 2000.