Solving Scheduling Problems with Setup Times and Alternative Resources

F.Focacci[1] and **P.Laborie**[2] and **W.Nuijten**[2]

[1] Dip. Ingegneria, Univ. Ferrara, Via Saragat, 41100 Ferrara, Italy
email: ffocacci@deis.unibo.it
[2] ILOG S.A., 9 rue de Verdun, BP 85, F-94253 Gentilly, France
email: {wnuijten,plaborie}@ilog.fr

Abstract

In this paper a general shop scheduling problem with sequence dependent setup times and alternative resources is considered, where optimization criteria are both makespan and sum of setup times. Two cooperative models for the problem based on Constraint Programming are proposed. The first is used to model the scheduling constraints, while the second is a multi-path model used for setup optimization. Integrating lower bounding techniques for the sum of setup times, the multi-path model performs propagation based on reduced cost fixing. A solution method based on a two phase algorithm is described, and a computational study is performed both on instances known from literature as on newly proposed instances. It is shown that the cooperation of the two models significantly improves performance. Although the aim of the paper is to study the problem including alternative resources, for several known instances without alternative resources, we were able to improve on the best known results.

Introduction

The scheduling problem studied in this paper is a general shop problem with sequence dependent setup times and alternative resources for activities. The setup time between two activities A_1 and A_2 is defined as the amount of time that must elapse between the end of A_1 and the start of A_2, when A_1 precedes A_2. If an activity can be scheduled on any one resource from a set S of resources, we say that S is the set of alternative resources for that activity. We consider two criteria as objective functions: makespan and sum of setup times. A large part of the motivation for this study was found in our experience with industrial applications. There we found that both sequence dependent setup times and alternative resources are very commonly encountered properties of scheduling problems. What's more, both properties are also often important in the sense that not considering them leads to unacceptable solutions. The importance of setup times and setup costs has also been investigated in several other studies. Allahverdi, Gupta, and Aldowaisan, in reviewing the research on scheduling involving setup considerations (Allahverdi, Gupta, & Aldowaisan 1998), discuss the importance of scheduling with sequence dependent setups in real world applications, and encourage researchers to work on the subject. Our experience with industrial applications also formed the basis for the motivation for considering both makespan and sum of setup times, and, more specifically of trying to find a schedule with a good makespan and a minimal sum of setup times. We chose to use Constraint Programming (CP) as it has been proven to be a very flexible framework to model and solve scheduling problems. Numerous industrial applications have been developed using constraint-based scheduling both because of its modeling capability as of the efficiency of specialized Operations Research (OR) algorithms that are embedded in global constraints. For more detailed discussions of constraint-based scheduling we refer to (Nuijten 1994; Baptiste, Le Pape, & Nuijten 1995; Caseau & Laburthe 1995; Le Pape & Baptiste 1996; Nuijten & Le Pape 1998; Beck 1999). We introduce two CP based models capturing two different aspects, i.e., the scheduling aspect and the multi-path aspect. The constraints representing the two models and a constraint linking them are described. The scheduling representation is essentially used to enforce feasibility, while the multi-path representation is used to effectively minimize the sum of setup times using OR techniques, i.e., lower bound calculation and reduced costs fixing. We propose a two phase procedure to solve the scheduling problem. In the first phase a good solution with respect to the makespan is found, after which in the second phase a local improvement method aims at minimizing the sum of setup times while maintaining a limit on the maximal makespan equal to the best makespan found during the first phase.

Problem Definition

We are given a set of n activities $A_1, ..., A_n$ and a set of m unary resources (resources with maximal capacity equal to one) $R_1, ..., R_m$. Each activity A_i has to be processed on a resource R_j for p_i time units. Resource R_j can be chosen within a given subset of the m resources. Sequence dependent setup times exist among activities. Given a setup time matrix S^k (square matrix of dimension n), s^k_{ij} represents the setup time between activities A_i and A_j if A_i and A_j are scheduled

sequentially on the same resource R_k. In such a case, $start_j \geq end_i + s^k_{ij}$. There may furthermore exist a setup time su^k_j before the first activity A_j can start on resource R_k and a teardown time td^k_i after the last activity A_i finishes on resource R_k. Activities may be linked by precedence relations $A_i \rightarrow A_j$. In this case activity A_j cannot start before the end of activity A_i. Constraints of the problem are therefore defined by the resource capacity, the transition times, the temporal relations, and the time bounds of the activities (release date and due date). The goal we will follow is to first find a schedule with the best possible makespan after which we will try to minimize the sum of setup times.

Related Work

A comprehensive review of the research on scheduling involving setup considerations was given in (Allahverdi, Gupta, & Aldowaisan 1998). The authors review the literature on scheduling problems with sequence dependent and sequence independent setup times, on single machine and parallel machines. They finally suggest directions for future research in the field. In this paper we follow some of these directions, i.e., emphasis on multi-machine scheduling problems, on multi criteria objectives, and on a generalized shop environment. An important reference for the work we propose is the paper of Brucker and Thiele (Brucker & O.Thiele 1996) where the authors propose a branch and bound algorithm for a scheduling problem with sequence dependent setup times. We generalize the problems described in this paper by considering alternative machines. Moreover, while for the authors the makespan is the only objective that is taken into account, we consider both makespan and sum of setup times. For general considerations on cost-based filtering used in the setup optimization we refer to (Focacci, Lodi, & Milano 1999a).

Models

Scheduling Model

The scheduling part of the problem is modeled by using ILOG Scheduler (Scheduler 1999). Each activity A_i is represented by using two variables being its start time $start_i$ and its end time end_i. These two variables are constrained by the relation $end_i - start_i = p_i$. Each resource R_k is represented by a unary resource.

A partition of the whole set of resources into alternative resource sets M_h is given. Each activity A_i is to be processed on a given alternative resource set M_h; it means that the activity will have to be executed on a resource R_k chosen among the resources belonging to M_h. $resource_i \in \{1, \ldots, m\}$ denotes the variable whose value represents the index k of the resource $R_k \in M_h$ that will be chosen for activity A_i. Setup times are represented in the scheduling model via a n square matrix S^k associated with each resource R_k. Precedence constraints and time bound constraints are posted on the variables $start_i$ and end_i of activities. As we will see in section Scheduling Constraints, the scheduling model allows the propagation of precedence, time bound, setup times and resource availability constraints over the variables of activities $start_i$, end_i and $resource_i$.

Path Model

We use a path model as a relaxation of the scheduling problem. We have a set of *start* nodes, a set of *internal* nodes, and a set of *end* nodes. Each internal node i represents an activity A_i. We are looking for m disjoint paths in the graph defined by these three sets. Each path represents a different resource. It starts in the start node of the resource, traverses a sequence of internal nodes, and ends in the end node of the resource. More precisely, let $I = \{0, 1, 2, \ldots n-1\}$ be a set of n nodes, $E = \{n, n+1, \ldots, n+m-1\}$, and $S = \{n+m, n+m+1, \ldots, n+2*m-1\}$ be two sets of m nodes. Nodes in I represent internal nodes, nodes in S, and in E represent start and end nodes respectively (see Figure 1). A global constraint $PathCst$ can be defined ensuring that m different paths $p_0, p_1, \ldots, p_m$ exists such that all internal nodes are visited exactly once by a path starting from a node in S, and ending into a node in E. Start nodes $n+m, n+m+1, \ldots, n+2*m-1$ belong respectively to paths $p_0, p_1, \ldots, p_m$. End nodes $n, n+1, \ldots, n+m-1$ belong respectively to paths $p_0, p_1, \ldots, p_m$. Moreover, sets P of possible paths can be associated to each internal node. We define three domain variables per node. Domain variables $Next_i$ and $Prev_i$ identify the nodes visited just after and just before node i respectively. Domain variables $Path_i$ identify the path the node belongs to. The domain of variables $Next_i$, and $Prev_i$ contains values $[0..n+2*m-1]$; the domain of variables $Path_i$ contains values $[0..m-1]$. Each start and end node has its path variable bound ($Path_n = [0], \ldots, Path_{n+m-1} = [m-1]; Path_{n+m} = [0], \ldots, Path_{n+2*m-1} = [m-1]$). In order to maintain a uniform treatment of all nodes inside the constraint, each start node $n+m+k$ has its $Prev_{n+m+k}$ variable bound to the corresponding end node ($Prev_{n+m+k} =$ [n+k]), and each end node $n+k$ has its $Next_{n+k}$ variable bound to the corresponding start node ($Next_{n+k}$ = [n+m+k]). A feasible solution satisfying the constraint $PathCst$ is an assignment of a different value to each next variable (the next node in the path) avoiding sub-tours (tours containing only internal nodes) such that

$$Next_i = [j] \leftrightarrow Prev_j = [i] \quad (1)$$

$$Next_i = [j] \Rightarrow Path_i = Path_j \quad (2)$$

A transition cost function can be associated to the path constraint, such that if node i is decided to be next to node j on a path k ($Next_i = [j], Path_i = Path_j = k$) a cost t^k_{ij} must be considered. In this case an optimal solution of the problem is the one minimizing $\sum_{i=0}^{n-1} t^{Path_i}_{i\, Next_i}$. If the transition cost function does not depend on the selected path, the path constraint defines a multiple travelling salesman problem (MTSP) on a digraph $G = (V, A)$ where $V = 0, 1, \ldots, n+2*m-1$ is the vertex set, and $A = (i, j) : i, j \in V$ is the arc

set. Cost c_{ij} is associated to each arc (i,j); $c_{ij} = t_{ij}$ if $j \in Next_i$; $c_{ij} = \infty$ otherwise. As said, the path model described represents a relaxation of the scheduling problem. If an internal node i has its next variable assigned to another internal node j, activity A_i directly precedes activity A_j; if an internal node i has its next variable assigned to an ending node $n+k$, activity A_i is the last activity scheduled on resource R_k. The transition cost function of the path model represent the transition time (setup time) among activities; therefore the minimization of $\sum_{i=0}^{n-1} t^{Path_i}_{i\,Next_i}$ in the path model corresponds to the minimization of the sum of setup times in the scheduling model.

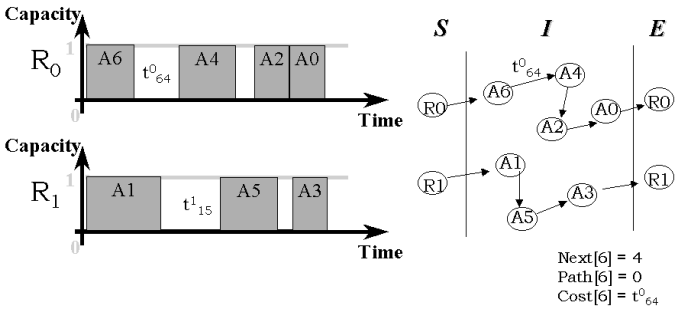

Figure 1: Models

In Figure 1 a schedule of 6 activities on 2 resources is shown with its correspondent path model. The scheduling model and the path model can be linked together and can cooperate for the solution of the problem by exploiting different views of the same problem.

Scheduling Constraints

In this section, we briefly describe the scheduling constraints that are used to perform propagation on the scheduling model.

Temporal Constraint The temporal constraints represent the precedences between activities given in the problem definition. The precedence constraint between two activities A_i and A_j is propagated as a constraint $end_i \leq start_j$. If a precedence graph is used (see section Precedence Graph Constraint), these precedence constraints are also taken into account by the precedence graph.

Disjunctive Constraint The disjunctive constraint aims at discovering new precedences by looking at pairs of activities that require the same unary resource. If A_i is an activity of the problem, we respectively denote $smin_i$, $smax_i$, $emin_i$ and $emax_i$ the earliest start time, latest start time, earliest end time and latest end time of activity A_i. Let A_i and A_j be two activities that require the same unary resource R_k. If $emin_j + s^k_{ji} > smax_i$, it means that activity A_j will not have enough time to execute before activity A_i. See an illustration on Figure 2 where we suppose that the setup time between A_1 and A_2 is 5. Thus, as both activities require the same unary resource R_k, A_i will have to be processed on R_k before A_j and the following domain reduction can be performed[1]: $smin_j = MAX(smin_j, emin_i + s^k_{ij})$ and $emax_i = MIN(emax_i, smax_j - s^k_{ij})$. On the example of the figure, it leads to a new earliest start time of 20 for activity A_2 and a new latest end time of 15 for activity A_1. Whenever the earliest end time

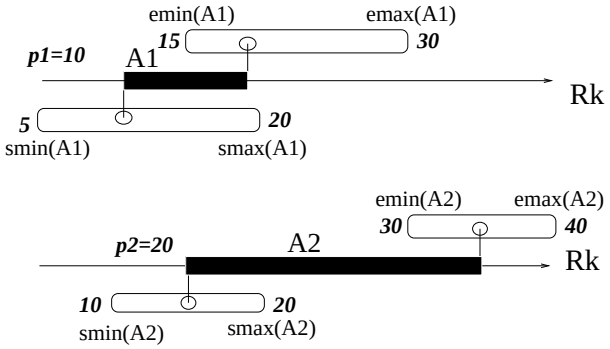

Figure 2: Disjunctive Constraint

or the latest start time of an activity A_i changes, the disjunctive constraint traverses the set of activities A_j that are to be processed on the same resource as A_i in order to perform this propagation.

Edge-Finding Constraint The edge-finding constraint is a constraint more powerful than the disjunctive constraint. It propagates the start and end time of one activity with respect to a subset of other activities. In general the edge-finding constraint can deduce that one activity A_i is to be scheduled before or after a set S of activities that are all to be scheduled on the same resource. In Figure 3 we give an example of such a deduction. For a more detailed description we refer to (Nuijten 1994). In the example of Figure 3, the sum of the processing times on resource R_k on the time interval $[0, 30)$ is $p_1 + p_2 + p_3 = 25$. Thus, on this resource not enough slack time exists to allow the processing of activity A (whose duration is 20) on the interval $[0, 30)$. Activity A must thus be processed on R_k after activities A_1, A_2, and A_3 and its new propagated earliest start time is 25. A similar reasoning allows the edge-finding constraint to restrict the domain of the possible end times of activities by proving that a given activity must be processed before a subset of activities.

Alternative Resource Constraint As seen in the scheduling model, each activity A_i may be processed on a resource R_k chosen within a given set of possible alternative resources $M_h = \{R_{h,1}, ..., R_{h,p}\}$. Alternative resources are propagated as if the activity A_i was split into p alternative activities $A_{i,k}$ where each activ-

[1] If a precedence graph is used (see section Precedence Graph Constraint), this domain reduction will be performed by the precedence graph constraint. In that case, the disjunctive constraint only adds the new precedence to the graph.

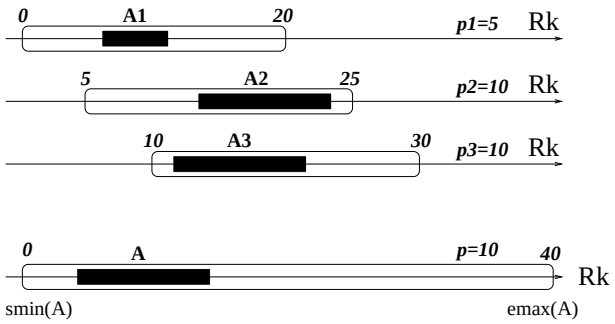

Figure 3: Edge-Finding Constraint

ity $A_{i,k}$ requires resource $R_{h,k}$ (see an illustration on Figure 4 for an activity A_i that must be processed either on R_1 or R_2). The alternative resource constraint maintains the constructive disjunction between the alternative activities $A_{i,k}$ that is, it ensures that:

- $smin_i = MIN_{k=1..p}\ smin(A_{i,k})$
- $smax_i = MAX_{k=1..p}\ smax(A_{i,k})$
- $emin_i = MIN_{k=1..p}\ emin(A_{i,k})$
- $emax_i = MAX_{k=1..p}\ emax(A_{i,k})$

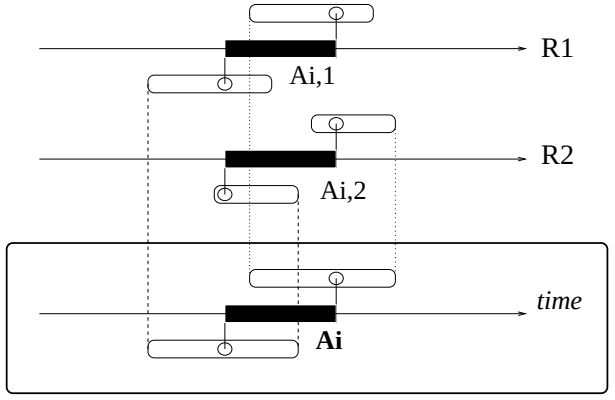

Figure 4: Alternative Resource Constraint

The scheduling constraints (disjunctive, edge-finding constraint) deduce new bounds for the alternative activities $A_{i,k}$ on the alternative resources R_k. Whenever the bounds of an activity $A_{i,k}$ turn out to be incoherent, the resource R_k is simply removed from the set of possible alternative resources for activity A_i. This is done by removing k from the possible values of the variable $resource_i$ that represents the resource on which activity A_i will be processed.

Path Optimization Constraint

In this section, we describe the cost-based domain filtering algorithms previously used in (Focacci, Lodi, & Milano 1999b) and (Focacci, Lodi, & Milano 1999a) for TSPs, TSPTW, and Matching Problems, and proposed as a general technique in (Focacci, Lodi, & Milano 1999a). The idea is to create a global constraint embedding a propagation algorithm aimed at removing those assignments from variable domains which do not improve the best solution found so far. Domain filtering is achieved by optimally solving a problem which is a relaxation of the original problem.

In this paper, we consider the Assignment Problem (AP) (Dell'Amico & Martello 1997) as a relaxation of the Path Model described (and consequently of the global scheduling problem). The AP is the graph theory problem of finding a set of *disjoint* sub-tours such that all the vertices in a graph are visited and the overall cost is minimized.

The Path Model (PM) looks for a set of m disjoint paths each of them starting from a start node and ending into the corresponding end node covering all nodes in a graph. Considering each end node connected to the correspondent start node, the Path Model looks, in other words, for a set of m disjoint tours each of them containing a start node. A correspondent AP can be formulated on the graph defined by the set of nodes in PM and the set of arcs (i, j) such that $j \in Next_i$. The cost on arc (i, j) is the minimal transition cost $min_k\{t^k_{ij}\}$. The optimal solution of the AP is obviously a lower bound on the optimal solution of the PM. The *primal-dual* algorithm described in (Carpaneto, Martello, & Toth 1988) provides an optimal integer solution for the AP with a $O(n^3)$ time complexity. The AP relaxation provides: the optimal AP solution, i.e., a variable assignment; the value of the optimal AP solution which is a lower bound LB on the original problem; a reduced cost matrix $\bar{c}$. Each $\bar{c}_{ij}$ estimates the additional cost to be added to LB if variable $Next_i$ is assigned to j. We have used these results to perform domain filtering and to define branching strategies. The lower bound value LB is trivially linked to the variable Z representing objective function of the sum of setup times through the constraint $LB \leq Z$. More interesting is the propagation based on reduced costs. Given the reduced cost matrix $\bar{c}$ of element $\bar{c}_{ij}$, it is known that $LB_{Next_i=j} = LB + \bar{c}_{ij}$ is a valid lower bound for the problem where $Next_i$ is assigned to j. Therefore we can impose:

$$\forall i, j\ LB_{Next_i=j} > Z_{max} \Rightarrow\ Next_i \neq j \qquad (3)$$

An improvement on the use of the reduced costs can be exploited as follows: we want to evaluate if value j could be removed from the domain of variable $Next_i$ on the basis of its estimated cost. Let $Next_i = k$ and $Next_l = j$ in the optimal AP solution. In order to assign $Next_i = j$, l and k must be re-assigned. The exact cost of this re-assignment can be calculated in $O(n^2)$, thus increasing the global complexity of the filtering algorithm. In (Focacci *et al.* 1998), two bounds on this cost have been proposed, whose calculation does not increase the total time complexity of the filtering algorithm which therefore remains $O(n^2)$. The events triggering this propagation are changes in the upper bound of the objective function variable Z and each change

in the problem variable domains (next, prev, and path variables). Note that the AP solution is recomputed only when the cost of an arc (i,j) that is part of the current AP solution increases its value over a certain threshold. The threshold T can be calculated as the minimum between the minimal reduced cost on row i and the minimal reduced cost on column j (excluding the zero reduced cost $\bar{c}_{ij}$).

$$T = min(min_{h \neq j}(\bar{c}_{ih}), min_{k \neq i}(\bar{c}_{kj})) \quad (4)$$

The AP recomputation is needed every time the removed value j from $Next_i$ belongs to the solution of the assignment problem (cost t_{ij} is set to infinite), and it may be needed when the domain of $Path_i$ increases the minimal cost t^*_{ij} that is to be paid to go from i to j in any of the remaining possible paths. In all other cases no recomputation is needed since an increase in cost of an arc that does not belong to the optimal solution does not change the optimal solution itself. The solution of the AP relaxation at the root node requires in the worst case $O(n^3)$, whereas each following AP recomputation due to domain reduction can be efficiently computed in $O(n^2)$ time, see (Carpaneto, Martello, & Toth 1988) for details. The reduced cost matrix is obtained without extra computational effort during the AP solution. Thus, the total time complexity of the filtering algorithm is $O(n^2)$. Reduced cost fixing appeared to be particularly suited for Constraint Programming. In fact, while reduced cost fixing is extensively used in OR framework, it is usually not exploited to trigger other constraints, but only in the following lower bound computation, i.e., the following branching node. When embedded in a CP framework, the reduced cost fixing produces domain reduction which usually triggers other problem constraints through shared variables.

Precedence Graph Constraint

We now need a way to link the path model and the scheduling model. This is done thanks to a precedence graph constraint. This constraint maintains for each resource R_k an extended precedence graph G_k that allows to represent and propagate temporal relations between pairs of activities on the resource as well as to dynamically compute the transitive closure of those relations (Laborie 1999). More precisely, G_k is a graph whose vertices are the alternative activities $A_{i,k}$ that may execute on resource R_k. A node $A_{i,k}$ is said to *surely contribute* if resource R_k is the only possible resource on which A_i can be processed. Otherwise, if activity A_i can also be processed on other resources, the node $A_{i,k}$ is said to *possibly contribute*. Two kind of edges are represented on G_k:

- A *precedence edge* between two alternative activities $A_{i,k} \rightarrow A_{j,k}$ means that if resource R_k is chosen for both activities A_i and A_j, then A_j will have to be processed after A_i on R_k.
- A *next edge* between two alternative activities $A_{i,k} \Rightarrow A_{j,k}$ means that if resource R_k is chosen for both activities A_i and A_j then, A_j will have to be processed directly after A_i on R_k. No activity may be processed on R_k between A_i and A_j.

The first role of the precedence graph is to incrementally maintain the closure of this graph when new edges or vertices are inserted, i.e., to deduce new edges given the ones already posted on the graph. The following two rules give a flavor of how this closure is computed[2]:

1. If $A_{i,k} \rightarrow A_{l,k} \rightarrow A_{j,k}$ and $A_{l,k}$ surely contributes then $A_{i,k} \rightarrow A_{j,k}$ (Transitive closure through contributor).
2. If $A_{l,k} \Rightarrow A_{i,k}$ and $A_{l,k} \rightarrow A_{j,k}$ and $A_{l,k}$ surely contributes then $A_{i,k} \rightarrow A_{j,k}$ (Next-Edge closure on the left).

As shown in Figure 5, new edges are automatically added on the precedence graph G_k by the scheduling constraints (precedence, disjunctive, edge-finding constraints) and by the path optimization constraint (whenever a variable $Next_i$ is bound a new Next-edge is added). Besides computing the incremental closure, the precedence graph also incrementally maintains the set of activities that are possibly next to a given activity $A_{i,k}$ given the current topology of G_k. It allows to effectively reduce the domain of the variables $Next_i$ and $Prev_i$ in the path model. Furthermore, the precedence graph constraint propagates the current set of precedence relations expressed on G_k on the start and end variables of activities.

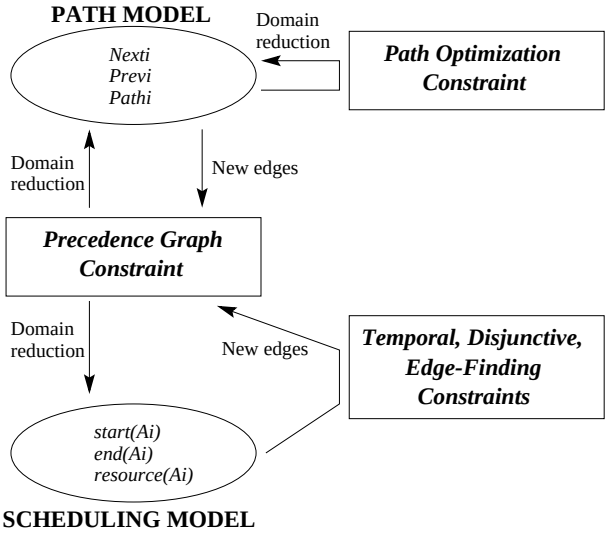

Figure 5: Architecture

Problem Solving

The problem is solved in two phases: during the first phase a *good* solution w.r.t. makespan is searched for. Let the best makespan found in this phase be m^*. In the second phase a constraint is added to the system imposing that any further solution will have a makespan

[2]For reasons of space, the set of rules we describe here is not complete. The set of rules for ensuring a complete closure contains 5 rules (Laborie 1999).

smaller or equal to m^*. Local improvement methods are used to minimize the sum of setup times.

First Phase Heuristic

A time-limited, incomplete branch and bound method is used to find solutions trying to minimize the makespan. At each node of the search tree we administer for each resource which activity has been scheduled last. The set B contains all these activities. By analyzing the precedence graph, we choose an activity A_i among the set of activities that are still unscheduled and can be next to one of the activities in B. We branch on the relative position of A_i forcing A_i to be either *next* or *successor but not next* of one activity in B. Among all activities that can be chosen we select the one having the earliest possible start time and, in case of ties, the one having the smallest latest end time. In the left branch of the tree, imposing A_i next to one activity in B, we also need to choose the resource assignment for A_i. If one or more resource assignments are feasible we heuristically choose one of them, otherwise we backtrack. We choose the resource assignment that allows to schedule the activity as early as possible, and in case of ties, the one which generates the smallest setup time.

Setup Optimization

In the setup optimization phase, given a solution having a makespan equal to m^*, and a total setup equal to s^*, we call for solutions having makespan less or equal to m^*, minimizing the sum of setup times. The improvement technique used is a time window based local optimization procedure. A time window $[TW_L^k, TW_U^k]$ defines a subproblem P_{TW^k} in the following way: in every resource, all activities on the left of the window are fixed (their start times and resource assignments are fixed); all activities on the right of the time window have their resource assignment fixed, and the sequence of the activities is also fixed (the variable *next* is fixed); all activities within the current window are completely free. On each subproblem a time limited branch and bound search is used to (possibly) find the optimal solution for P_{TW^k}. The branch and bound technique used to minimize the sum of setup times can effectively exploit the optimization constraint to reduce the search space, and eventually guide the search. In fact, the computational results will show that when the optimization constraint is used most subproblems are quickly solved up to optimality. Two different methods have been used to select

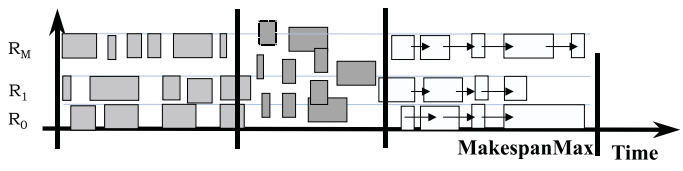

Figure 6: Setup optimization

the current window: a simple gliding window method and a lower bound based method.

Gliding Window When the gliding window method is used, given a fixed size of the window W_{size}, and a window offset W_{delta}, we start the setup optimization at P_{TW^0} defined by window $[0, W_{size}]$, we optimize the problem, then we move the right and left bound of the window of W_{delta} to the left.

$$P_{TW^{k+1}} \leftarrow [TW_L^k + W_{delta}, TW_U^k + W_{delta}] \qquad (5)$$

We repeat this until the end of the schedule is reached. At the end of each loop, the window size and offset can eventually be modified and another loop can be performed.

LB-based Window Selection The method described here is based on the idea to work first on the part of the schedule where we can hope to obtain the highest improvement. For a given subproblem P_{TW^k}, defined by window $[TW_L^k, TW_U^k]$ we can calculate the expected improvement on the objective function E_{TW^k} as the difference between the current sum of setup times in that window, and the lower bound calculated in that window. After subproblem P_{TW^k} is defined, variable Z identifying the sum of setup times contains the information of the lower bound calculated by the optimization constraint together with the precedence graph constraint and the scheduling constraints. Therefore if s^* is the total setup value of the current best solution found, E_{TW^k} is simply equal to $s^* - Z_{min}$. In the LB-based window selection we first calculate the expected improvement on the objective function E_{TW^k} for a certain number of subproblems (depending on parameters similar to W_{size} and W_{delta}), and then sort the subproblems in descending order of E_{TW^k}. All subproblems that may lead to an improvement of the objective function are labeled as improvable. We run the branch and bound algorithm on the first ranked subproblem that can lead to an improvement, and change the label of the window. If a better solution is found, the current solution is updated, and the values E_{TW^k} of all windows on the right of the modified one are recalculated since they may have been changed by the new solution. Also, the labels of the windows on the right are updated. The windows are then re-sort and the procedure is repeated until no window exists that is labeled as improvable.

Computational Results

In the computational results we primarily try to show that the integration of lower bounding OR techniques in Constraint-Based Scheduling can improve performances both in terms of computation time and quality of solutions. We show that the large neighborhood defined by a time window containing between 30 and 60 activities can very effectively be solved by means of the interaction among the optimization constraint based on a lower bound calculation and reduced cost fixing, the precedence graph, and the scheduling constraints available in ILOG Scheduler. We have tested the proposed models on real world applications, moreover, the definition of the problem itself, and of the objective functions is a direct consequence of the real applications considered.

In the following we will describe the generation process of the instances that we used to be able to compare our results with, hopefully, different approaches that could be presented. We will finally show the computational results.

Instance generation

Following the computational studies in (Brucker & O.Thiele 1996), we run experiments on open-shop, general-shop and job-shop problems. We use the instances of Brucker and Thiele (available on the Web at http://www.mathematik.uni-osnabrueck.de/research/OR) to test the approach on scheduling problems with setup times without resource alternatives. We then duplicate and triplicate these instances to generate scheduling problems with setup times and alternative resources. In order to generate an instance with the alternative choice of k resources, for each activity and each resource in the original instance k activities and k resources are created. If in the original instance activity A_i requires resource R_j, in the k-multiplied instance each one of the A_{ih} identical activities requires one out of the k identical resources R_{jh}. The temporal constraints among activities are also duplicated such that if a temporal constraint exists in the original instance $A_i \rightarrow A_j$, the set of temporal constraints $A_{ih} \rightarrow A_{jh}$ exist in new instance. For all instances without alternative resources we can qualitatively compare our results with the results published in (Brucker & O.Thiele 1996). Nevertheless, a real comparison cannot be done since in (Brucker & O.Thiele 1996) the objective is the minimization of the makespan, while we want to minimize the sum of setup times in a problem constrained by a maximal makespan. The open-shop problems considered contain, in the non-alternative instances, 8 machines and 8 jobs (16 machine and 16 jobs for the 2-alternatives instances etc..). The general-shop problems considered also contain, in the non-alternative instances, 8 machines and 8 jobs, and derives from the open-shop problems with the addition of the temporal constraints described in (Brucker & O.Thiele 1996). The job-shop problems considered contain 5 machines and 20 jobs in the non-alternative instances.

Results

Tables 1 to 3 report results on open-shop, general-shop, and job-shop instances. Table 1 reports results for the original instances from (Brucker & O.Thiele 1996). Table 2 and 3 report results for the 2 and 3-multiplied instances respectively, generated as described above, as alternative resources problems. For each problem we report the results obtained by the first solution phase, and the setup optimization phase in terms of sum of setup times (*su*), and makespan (*mks*). We finally report the results published in (Brucker & O.Thiele 1996) in terms of makespan for all the instances without alternative of resources. All tests run on a Pentium II 300 MHz. The results published in (Brucker & O.Thiele 1996) were obtained on a Sun 4/20 workstation where a time limit of 5 hours was set for open-shop and general-shop problems, and a time limit of 2 hours was set for the job-shop problems.

Column *FirstSol* reports results in terms of makespan and sum of setup times of the best solutions obtained using the time limited branch and bound strategy described in section First Phase Heuristics. The time limit given was 60 seconds, and a Limited Discrepancy Search tree exploration was used, see (Harvey & Ginsberg 1995; Perron 1999). The solution obtained after this phase is thought to be a good solution w.r.t. makespan minimization. For example, for all the problems without alternative resources the makespan obtained is very close to the best known solution published in (Brucker & O.Thiele 1996). In half of the instances considered, the makespan found in the first solution phase improves the best known published in (Brucker & O.Thiele 1996). These results were used as starting point for setup optimization.

In the setup optimization phase we fixed an initial window size of 30 activities (i.e. each subproblem has 30 completely free activities), and we used a 5 seconds time limited branch and bound algorithm to minimize the sum of setup times in each subproblem. In order to compare the results obtained with and without the optimization constraint, we used always the same very simple branching strategy: we choose the variable *next* with the smallest domain and we branch on its possible values starting from the one generating the smallest setup time. Given an initial window size, the setup optimization methods (columns *noLB GW*, *LB GW*, *LB Rank*) are called until a local minimum is reached, then the window size is increased 20% (e.g. from 30 free activities to 36 free activities), and the procedures are repeated until a global time limit is reached. Column *noLB GW* and column *LB GW* report the results obtained by the gliding window method described in section Gliding Window. The algorithm used for column *noLB GW* does not calculate the lower bound on the sum of setup times, while the algorithm used for column *LB GW* makes full usage of the pruning based of the lower bound calculation and the reduced cost fixing. Column *LB Rank* reports the results obtained by the method described in section LB-based Window Selection.

For the open-shop and general-shop instances of Table 1 (containing 64 activities each) the global time limit used is 30 seconds. For the job-shop instances of Table 1 (containing 100 activities each), for the open-shop, and general-shop instances of Table 2 (containing 128 activities each) the global time limit used is 60 seconds. For the job-shop instances of Table 2 (containing 200 activities each), for the open-shop, and general-shop instances of Table 3 (containing 192 activities each) the global time limit used was 120 seconds. For the job-shop instances of Table 3 (containing 300 activities each) the global time limit used is 240 seconds.

	FirstSol		noLB GW		LB GW		LB Rank		BT96
open-shop									
	su	mks	su	mks	su	mks	su	mks	mks
TAIBS81	2680	942	1740	928	1620	919	1480*	936	914*
TAIBS85	3480	1113	2180	985	1280*	1108	1280*	1108	899*
TAIS81	1460	699	980	693	890*	690*	980	693	713
TAIS85	1850	755	1260	748	790*	748	850	754	747*
general-shop									
	su	mks	su	mks	su	mks	su	mks	mks
TAIBGS81	1680	763	1410*	763	1410*	763	1470	759*	837
TAIBGS85	2010	869	1150	862	870*	867	870*	867	762*
TAIGS81	1510	734*	1190	734*	1150*	734*	1190	734*	858
TAIGS85	1540	749	1210	745*	1010*	747	1160	747	783
job-shop									
	su	mks	su	mks	su	mks	su	mks	mks
T2-PS12	1710	1450	1640	1450	1640	1450	1530*	1448*	1528
T2-PS13	1930	1669	1640	1667	1640	1667	1430*	1658	1549*
T2-PSS12	1480	1367	1300	1367	1300	1367	1220*	1362*	1384
T2-PSS12	1290	1522	1220	1522	1140*	1522	1220	1518	1463*

Table 1: Original instances from Brucker & Thiele 1996.

	FirstSol		noLB GW		LB GW		LB Rank	
open-shop								
	su	mks	su	mks	su	mks	su	mks
TAIBS81	3920	908	3760	903	2760*	904	2840	905
TAIBS85	4520	942	4260	940	2580	939	2380*	942
TAIS81	2220	723	2060	723	1540*	723	1590	723
TAIS85	2280	690	2110	689	1730*	690	1950	689
general-shop								
	su	mks	su	mks	su	mks	su	mks
TAIBGS81	2220	1023	2140	1023	1270*	1008	1490	1017
TAIBGS85	2640	1031	2350	1019	1300	1020	1150*	1026
TAIGS81	2510	766	2430	764	1900	766	1720*	756
TAIGS85	2490	748	2450	748	1810	743	1710*	748
job-shop								
	su	mks	su	mks	su	mks	su	mks
T2-PS12	3410	1562	2980	1537	2330*	1552	2510	1551
T2-PS13	2890	1593	2670	1593	2270	1584	2240*	1593
T2-PSS12	2090	1515	1820	1479	1610	1505	1540*	1515
T2-PSS12	2120	1578	1720	1576	1520*	1574	1590	1545

Table 2: Instances with alternative of two resources.

	FirstSol		noLB GW		LB GW		LB Rank	
open-shop								
	su	mks	su	mks	su	mks	su	mks
TAIBS81	4780	1002	4380	999	3320*	999	3520	986
TAIBS85	5320	875	5280	875	4180	870	4160*	865
TAIS81	2910	802	2440	802	2190	800	2090*	802
TAIS85	2660	758	2540	758	2020	755	1690*	757
general-shop								
	su	mks	su	mks	su	mks	su	mks
TAIBGS81	2230	1083	2140	1067	1380*	1079	1540	1083
TAIBGS85	2240	1280	2080	1280	1550	1268	1410*	1268
TAIGS81	2470	887	2430	887	1740	887	1670*	885
TAIGS85	2900	789	2710	789	1760*	789	1850	787
job-shop								
	su	mks	su	mks	su	mks	su	mks
T2-PS12	2870	1593	2740	1593	2640	1587	2210*	1585
T2-PS13	2600	1585	2600	1585	2400*	1585	2500	1585
T2-PSS12	2500	1455	2360	1455	2240*	1455	2290	1455
T2-PSS12	2100	1562	1850	1562	1770	1562	1730*	1562

Table 3: Instances with alternative of three resources.

When the optimization constraint is used in collaboration with scheduling propagation algorithms the solutions obtained are always a lot better than the ones obtained without the optimization constraint. The improvement in the solution quality is particularly important for problems with two and three alternative resources. Problems without alternative resources are easier and even without the optimization constraint, in each window, the local optimal solution can often be found. Nevertheless, even in these cases, when the optimization constraint is used the subproblems are solved up to optimality in a shorter time.

More difficult is the comparison between the LB-based Window Selection and the Gliding Window method since the best solutions are equally distributed between the two methods. We cannot at this point claim that the LB-based Window Selection method outperforms the simpler Gliding Window method. Indeed, if the scheduling problem is small enough to allow one or several complete gliding window loops, the LB-based method may loose some interest: if all windows are considered, the order in which they are solved may not be too important. On the other hand, for very large problems a complete gliding window loop may not be possible within the CPU time available. In such a case, a fast evaluation of the most promising area for improvement may play an important role. Further analysis of the relative advantages of the two methods will be subject of future work. In Figures 7-9, the plot of Table 1-3 is reported; the x-ax represents the problem instance, while the ration between the final sum of setup times, and the sum of setup times of the first solution is reported on the y-ax.

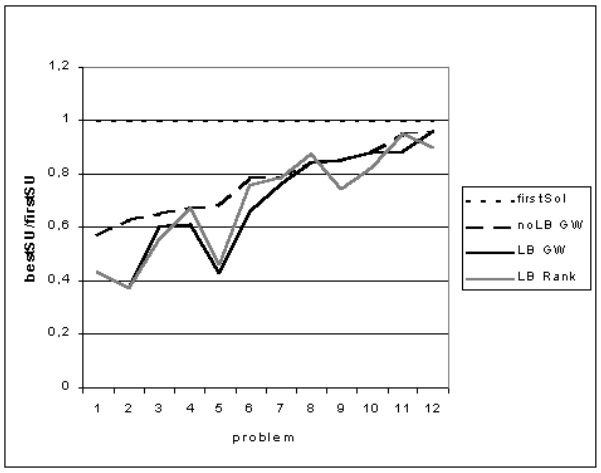

Figure 7: Instances without alternative

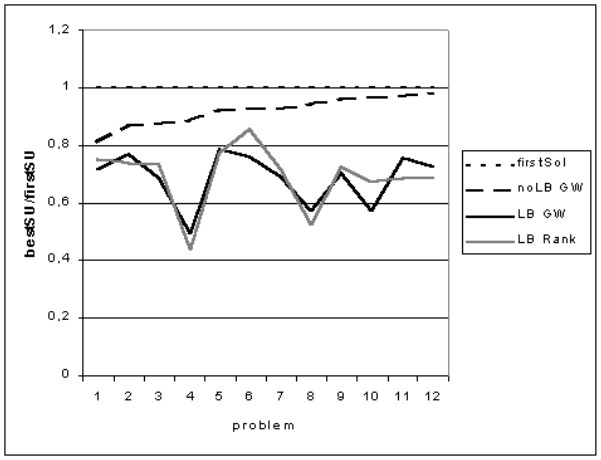

Figure 8: Instances with two resources alternative

It is interesting to look in more details at the results obtained by the optimization constraint compared to the ones obtained without the lower bound calculation. In Table 4 we report, for each type of problem, the

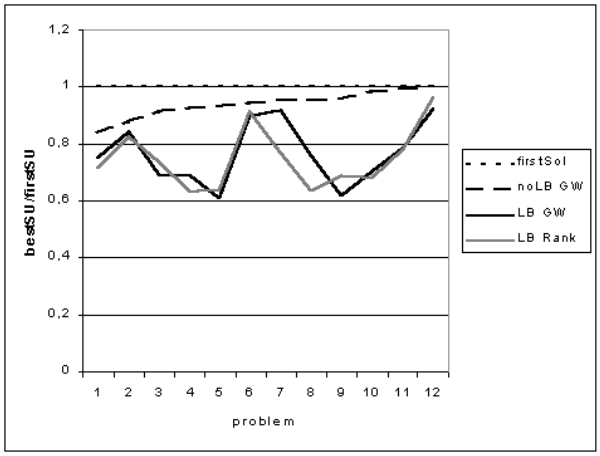

Figure 9: Instances with three resources alternative

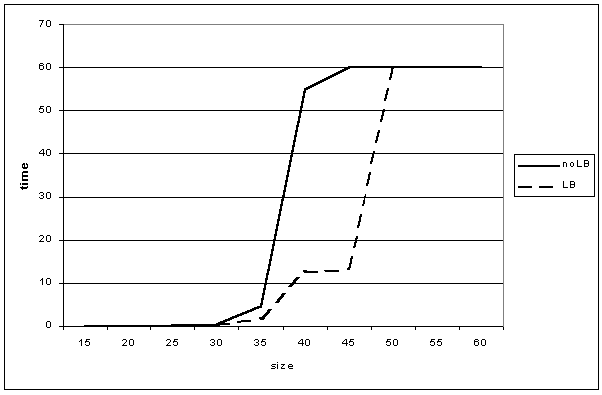

Figure 10: Instances without alternative

average number of subproblems created, the average number of subproblems solved up to optimality (in percentage), and the average time spent in each window, when one single gliding window loop is performed. We recall that the time limit for each window is 5 seconds, therefore whenever the average time spent per window is close to 5 seconds it means that most windows could not be solved up to optimality.

	win	opt	time	win	opt	time	win	opt	time
	1 instance			2 instances			3 instances		
	open-shop								
noLB	3	50%	2,6	7	21%	4,2	10,7	9%	4,8
LB	3	100%	0,2	7	100%	0,3	10,7	90%	0,9
	general-shop								
noLB	3	83%	1,4	6,2	20%	4,4	9,2	13%	4,7
LB	3	100%	0,3	6,2	84%	1,3	9,2	94%	0,6
	job-shop								
noLB	5,2	80%	1,7	12	8%	4,9	18,3	3%	5
LB	5,2	90%	1,1	12	56%	3	18,3	73%	2,3

Table 4: Windows statistics

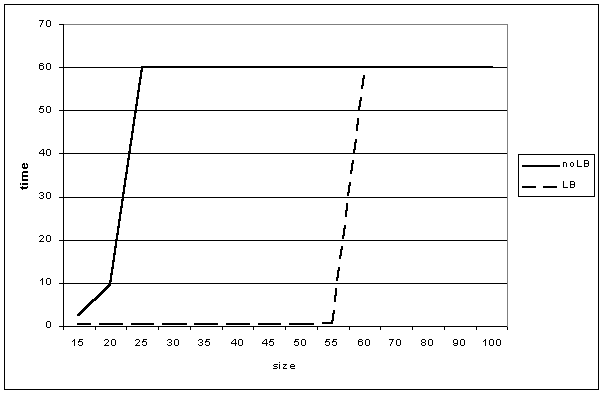

Figure 11: Instances with three resources alternative

We can see that when the optimization constraint is used we can solve, on average, 80% of the subproblems up to optimality (with the proof of optimality) within the 5 seconds allocated. On the other hand, when the optimization constraint is not used, the percentage of subproblems solved up to optimality quickly drops from an average of 70% for the problems without alternative resources to an average of less than 10% for the problems with alternatives of three resources. Moreover, the average time spent on each window when the optimization constraint is used always remains very small. Figure 10 and 11 compare the performance of solving (and proving optimality) of a setup minimization problem with and without the LB calculation for increasing problem size (in number of activities). Instances with up to 60 activities were easily solved within one minute.

One small remark on the fact that although the aim of the methods proposed is to study the problem including alternative resources, for several known instances without alternative resources, we were able to improve on the best known results.

Finally, some tests were run on small instances without alternative of resources taken form (Brucker & O.Thiele 1996). Open-shop and general-shop problems contain 4 machines and 4 jobs, while job-shop contain 5 machine and 10 jobs. For each problem two results are reported in Table 5. Column BestMk - BestSu reports the results obtained by optimizing the makespan in a first phase, and the sum of setup times in a second phase where the makespan was limited by the best value found in the first phase. Column BestSu - BestMk reports the results obtained by optimizing the sum of setup times in a first phase, and the makespan in a second phase where the sum of setup times was limited by the value found in the first phase.

open-shop						
	BestMk - BestSu			BestSu - BestMk		
	su	mks	time	su	mks	time
TAIBS01	380	306	0.38	320	384	2.58
TAIBS05	440	395	0.22	320	563	0.72
TAIS01	190	249	0.28	160	344	1.93
TAIS05	220	348	0.27	160	523	0.33
general-shop						
	BestMk - BestSu			BestSu - BestMk		
	su	mks	time	su	mks	time
TAIBGS01	280	322	0.16	160	527	0.06
TAIBGS05	280	491	0.33	160	747	0.06
TAIGS01	230	285	0.44	160	362	0.39
TAIGS05	300	384	0.22	160	546	0.16
job-shop						
	BestMk - BestSu			BestSu - BestMk		
	su	mks	time	su	mks	time
T2-PS01	710	798	-	250	2368	-
T2-PS02	630	784	88,42	250	2221	-
T2-PS03	550	749	144,2	250	1932	-
T2-PS04	670	730	388,31	250	1665	-
T2-PS05	710	691	30,43	250	1899	-

Table 5: Results on small instances.

In all cases where a computation time is reported, the optimal solution could be proven in both phases. For example, in problem TAIGS05 the optimal makespan is 384, and given such a makespan, the optimal sum of setup times is 300; on the other hand, the optimal sum of setup times also for problem TAIGS05 is 160, and given a limit on the sum of setup times equal to 160, the optimal makespan is 546. Where the time is not reported, optimality could not be proven within 30 minutes.

Two consequences can be taken from these results: the first consequence is that the minimization of only one objective between makespan and sum of setup times may generate poor quality solutions for the other objective. For this reason we think it is necessary, in practice, to consider multi criteria objectives. The second consequence is that the algorithm proposed, for small problems, is able to fix any limit for one objective and find the optimal solution for the other objective. Therefore the method could be used to optimize any combination of makespan and sum of setup times, and to find a set of pareto-optimal solutions.

Conclusion and Future Work

A general scheduling problem with a multi criteria objective function was defined which, to our experience, is of great practical interest. The problem was modeled using a CP approach based on ILOG Solver and Scheduler. A multi-path model was defined to take care of the sequence dependent setup view of the problem. We integrated OR lower bounding techniques and reduced cost fixing in the multi-path constraint in order to effectively prune the search space. A large neighborhood for setup optimization was proposed and we showed that the local optimal solution within the neighborhood can effectively be reached. We generated new problem instances to test the described approach. The computational results show that the cooperation between the scheduling and multi-path model can effectively be used to minimize the sum of setup times while maintaining the makespan constrained to be under a given threshold. Although the aim of the paper is to study the problem including alternative resources, for several known instances without alternative resources, we were able to improve on the best known results. We plan to extend our approach in several directions. We are working on the definition of several different neighborhoods, and are experimenting on the combination of them. Moreover, the use of the optimization constraint could also be exploited for generating more sophisticated branching strategies and heuristics.

References

Allahverdi, A.; Gupta, J.; and Aldowaisan, T. 1998. A review of scheduling research involving setup consideration. *Omega* forthcoming.

Baptiste, P.; Le Pape, C.; and Nuijten, W. 1995. Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proc. 1st International Joint Workshop on Artificial Intelligence and Operations Research.*

Beck, J. C. 1999. *Texture measurements as a basis for heuristic commitment techniques in constraint-directed scheduling.* Ph.D. Dissertation, University of Toronto.

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

Carpaneto, G.; Martello, S.; and Toth, P. 1988. Algorithms and codes for the assignment problem. *Annals of Operations Research* 13:193–223.

Caseau, Y., and Laburthe, F. 1995. Disjunctive scheduling with task intervals. Technical report, Ecole Normale Superieure.

Dell'Amico, M., and Martello, S. 1997. Linear assignment. In Dell'Amico, M.; Maffioli, F.; and Martello, S., eds., *Annotated Bibliographies in Combinatorial Optimization.* Wiley.

Focacci, F.; Lodi, A.; Milano, M.; and Vigo, D. 1998. Solving TSP through the integration of OR and CP techniques. *Proc. CP98 Workshop on Large Scale Combinatorial Optimisation and Constraints.*

Focacci, F.; Lodi, A.; and Milano, M. 1999a. Cost-based domain filtering. *Proc. International Conference on Principles and Practice of Constraint Programming CP99.*

Focacci, F.; Lodi, A.; and Milano, M. 1999b. Solving tsp with time windows with constraints. In *ICLP'99 International Conference on Logic Programming.*

Harvey, W., and Ginsberg, M. 1995. Limited discrepancy search. *Proc. IJCAI95.*

Laborie, P. 1999. Modal precedence graphs and their usage in ILOG Scheduler. Technical Report OIR-1999-1, ILOG.

Le Pape, C., and Baptiste, P. 1996. Constraint propagation techniques for disjunctive scheduling: The preemptive case. In *Proc. 12th European Conference on Artificial Intelligence.*

Nuijten, W., and Le Pape, C. 1998. Constraint-based job shop scheduling with ILOG SCHEDULER. *Journal of Heuristics* 3:271–286.

Nuijten, W. 1994. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach.* Ph.D. Dissertation, Eindhoven University of Technology.

Perron, L. 1999. Integration into constraint programming and parallelization of or/ai search methods. In *CP-AI-OR'99 Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems.*

Scheduler. 1999. *ILOG Scheduler 4.4 User's Manual and Reference Manual.* ILOG, S.A.