DEPARTMENT OF OPERATIONS RESEARCH
UNIVERSITY OF AARHUS

Working Paper no. 2004 / 2

Solving a large-scale precedence constrained scheduling problem with elastic jobs using tabu search

Christian R. Pedersen, Rasmus V. Rasmussen
and Kim A. Andersen

ISSN 1600-8987

Department of Mathematical Sciences
Telephone: +45 8942 1111
E-mail: institut@imf.au.dk

Building 530, Ny Munkegade
DK-8000 Aarhus C, Denmark
URL: www.imf.au.dk

Solving a large-scale precedence constrained scheduling problem with elastic jobs using tabu search

Christian R. Pedersen [a,*], Rasmus V. Rasmussen [a,*],

[a] *Department of Operations Research, University of Aarhus, Ny Munkegade, Building 530, 8000 Aarhus C, Denmark*

Kim A. Andersen [b]

[b] *Department of Accounting, Finance and Logistics, Aarhus School of Business, Fuglesangs Allé 4, 8210 Aarhus V, Denmark*

Abstract

This paper presents a solution method for minimizing makespan of a practical large-scale scheduling problem with elastic jobs. The jobs are processed on three servers and restricted by precedence constraints, time windows and capacity limitations. We derive a new method for approximating the server exploitation of the elastic jobs and solve the problem using a tabu search procedure. Finding an initial feasible solution is in general $\mathcal{NP}$-complete, but the tabu search procedure includes a specialized heuristic for solving this problem. The solution method has proven to be very efficient and leads to a significant decrease in makespan compared to the strategy currently implemented.

Key words: Large-scale scheduling, elastic jobs, precedence constraints, practical application, tabu search.

1 Introduction

This paper focuses on a specific problem provided to us by the Danish telecommunications net operator, Sonofon. By the end of each day a rather large number of jobs (End-of-Day jobs) have to be processed on exactly one particular of

* Corresponding authors. Tel.: (+45) 89423536; fax: (+45) 86131769.
Email addresses: roed@imf.au.dk (Christian R. Pedersen), vinther@imf.au.dk (Rasmus V. Rasmussen), kia@asb.dk (Kim A. Andersen).

three available servers. The objective is to schedule the jobs on the machines in order to minimize makespan. This task is complicated by the fact that a large number of precedence constraints among the jobs must be fulfilled, that time windows must be obeyed and that capacity limitations must be respected. In addition, the jobs are *elastic* which means that the duration of a particular job depends on the capacity assigned to the job. Elasticity of jobs complicates the problem considerably and has to the best of our knowledge not yet been considered in large-scale scheduling.

The applications of scheduling problems are wide-spread, and hence a considerable amount of promising research has been devoted to such problems both within the operations research literature and the computer science literature. Especially during the past decade algorithms merging operations research techniques and constraint programming (CP) have proved efficient as exact solution methods for solving scheduling problems. Among a number of interesting CP contributions to small- or medium-scaled scheduling problems we mention the work by Jain and Grossmann [1], Hooker and Ottoson [2], Hooker [3] and Baptiste *et al.* [4,5]. For large-scale problems, meta heuristics in particular have shown promising results.

One classical meta heuristic that has been successfully applied to scheduling problems is tabu search, due to Glover [6] and Glover and Laguna [7]. The papers on tabu search are numerous, but let us for brevity only mention a few which all appeared recently and consider scheduling problems. Grabowski and Wodecki [8] consider large-scale flow shop problems with makespan criterion and develop a very fast tabu search heuristic focusing on a lower bound for the makespan instead of the exact makespan value. Ferland *et al.* [9] consider a practical problem of scheduling internships for physician students and propose several variants of tabu search procedures. The last three papers all consider the problem of scheduling a number of jobs to a set of heterogeneous machines under precedence constraints, with the objective of minimizing makespan. In Porto *et al.* [10] a parallel Tabu Search heuristic is developed and proved superior to a widely used greedy heuristic for the problem. In Chekuri and Bender [11] a new approximation algorithm is presented, but unfortunately, no computational results are reported. Finally, in Mansini *et al.* [12] jobs with up to three predecessors each are considered among groups of jobs requiring the same set of machines. The problem is formulated as a graph-theoretical problem. In the paper a number of approximation results are provided, but no computational experience is reported.

Clearly, the vast solution space and the complexity of the present problem called for a heuristic procedure. Due to the high flexibility of tabu search and its promising results with scheduling problems, we chose that method.

The contributions of this paper can be summarized as follows:

- We present a special designed heuristic based on tabu search to solve a large-scale practical problem provided to us by a large Danish telecommunications net operator, Sonofon. Today Sonofon faces the problem that the average completion time exceeds the deadline by 41 minutes. This means that with the existing scheduling strategy new hardware needs to be purchased in order to keep satisfying the given requirements. This paper shows that the existing hardware is, in fact, sufficient to complete the jobs in time and indeed spare capacity is available, when a good schedule is chosen.
- The algorithm is capable of handling large-scale scheduling problems with precedence constraints among jobs and time windows, and a new approximate method for scheduling elastic jobs is developed.
- A heuristic procedure for obtaining an initial feasible solution is provided. This proves to work very well on the specific application which cannot be solved by traditional IP/CP code.
- The solution method provides a significant improvement in makespan compared to the strategy currently implemented by Sonofon, and an improvement of 25 percent to the current solution is reported within 17 minutes of computation time. Sonofon expects to use the obtained solution in the future.
- The algorithm quickly finds a good solution and can be aborted at any time. Therefore as long as the jobs are known just prior to the actual scheduling process our algorithm is capable of producing a good feasible schedule.

The remaining part of the paper is organized as follows. In Section 2, we present the practical problem offered by Sonofon, followed by the derivation of a hybrid IP/CP model. In Section 3, we give a thorough introduction to the developed tabu search heuristic, and computational results are provided in Section 4.

2 Problem formulation

The Danish telecommunications net operator, Sonofon, faces a three-machine scheduling problem, with 346 End-of-Day jobs (EOD). Each job is dedicated to a particular server, it cannot be shared among the servers, and it has to be processed without *preemption* [1] .

The scheduling time horizon runs from 7.00 pm to 8.00 am, and each job receives a time window in which it should be processed. Unfortunately, the time windows are wide, leaving numerous feasible starting times for each job. Since most scheduling tools applying Constraint Programming relies heavily on propagation techniques, the wide time windows have a negative influence

[1] Preemption means that jobs can be interrupted during processing.

on the performance of such scheduling packages. The time windows will be explored further in Section 3.1. Since the servers immediately after completing the EOD-jobs are assigned to other operations, the objective will be to minimize makespan.

Because of interrelations between jobs, a number of precedence constraints must be fulfilled. It might occur that a job need information from a database to which another job (a predecessor) has written earlier.

In a real-world application each job can execute with varying capacity consumption during its runtime, as illustrated by job 1 in Figure 1(b). However, due to limitations of server exploitation, we can assume that each job has an upper bound of capacity consumption. In Figure 1(a) we have illustrated a situation in which three jobs are placed at a machine to start processing at time 0. Each of the three jobs is assumed to have a maximal capacity consumption of 15 units, and the machine has capacity 30. Since all the jobs are scheduled to start at time 0, they must share the available capacity, as shown in Figure 1(b). Observe that in Figures 1(a) and (b) the two corresponding boxes for a job have the same area. This would be an incorrect representation of a real-world application, since in general (duration × capacity) increases with decreasing capacity, due to lost server efficiency from *swapping* [2]. This fact is represented by the inclusion of the shaded area in Figure 1(c).

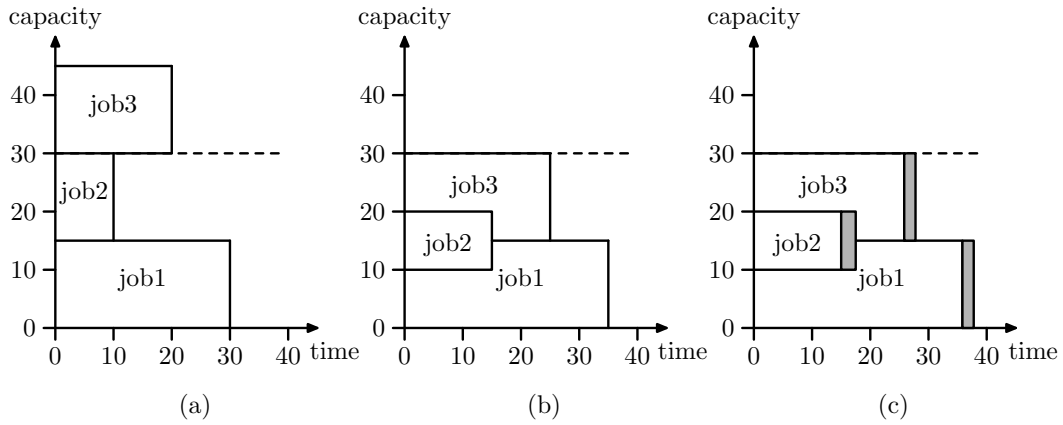

Figure 1. (duration × capacity) increases with decreasing capacity.

In our setup we shall assume that the capacity consumption for a job remains constant during its runtime. Opposed to other literature on large-scale scheduling we do not restrict time and capacity consumption to be given beforehand. Instead we assume that jobs are elastic, and hence allow the time and capacity consumption to be found during the optimization process. We deal with the non-linear functionality between time and capacity consumption by a rough

[2] Swapping or *trashing* means time being spent for reading jobs into and out of the temporary memory, not processing any job.

approximation representing each job as a choice between three boxes, (see Figure 2).

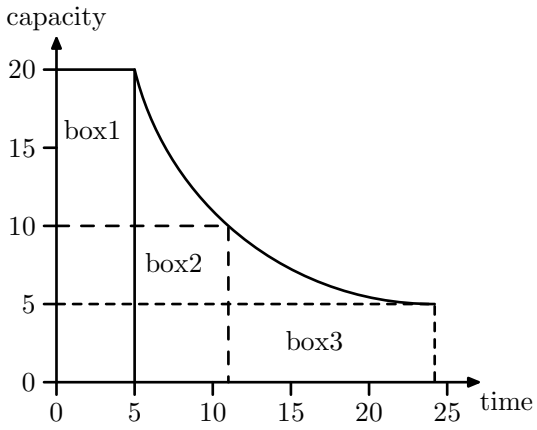

Figure 2. Three representations of a job.

The dimensions of the boxes for a given job j are explained in Table 1, where cap_j ($time_j$) corresponds to the capacity (duration) for the job-box having the least capacity consumption (and hence the longest duration) [3]. The second and third column gives the capacity and time consumption for a given box, and the last column gives (capacity consumption $\times$ duration). Notice that, by a 50% decrease in capacity, (capacity consumption $\times$ duration) increases by 10%. This trade-off between capacity assigned to a particular job and its duration was determined in correspondence with Sonofon and reflects the specific problem rather closely.

Table 1
Dimensions of boxes representing job j.

l	cap_{jl}	$time_{jl}$	$(cap_{jl} \times time_{jl})$
1	$4 \cdot cap_j$	$25/121 \cdot time_j$	$100/121 \cdot cap_j \cdot time_j$
2	$2 \cdot cap_j$	$5/11 \cdot time_j$	$10/11 \cdot cap_j \cdot time_j$
3	cap_j	$time_j$	$cap_j \cdot time_j$

Since the representation of scheduling problems is greatly simplified using the terminology from constraint programming, we too, shall adapt such a notation. Hence for our problem we derive a hybrid IP/CP model which is to be solved by a heuristic procedure, more specifically by a tabu search algorithm. Let us

[3] Time constitutes an average longest runtime provided by Sonofon from historical data.

introduce the following notation,

$M = \{1, 2, 3\}$ - Machines
$J = \{1, \ldots, n\}$ - Jobs
$P = \{(j, k) |$ job j shall precede job $k\}$ - Precedence constraints
$M_m = \{j |$ job j shall be processed on machine $m\}$ - Job-machine constraints
$L = \{1, 2, 3\}$ - Boxes for each job

For each job j, we introduce the four variables:

$j.start$ - Starting time
$j.duration$ - Duration
$j.end$ - Completion time
$j.capacity$ - Capacity consumption

connected by the implicit constraint $j.start + j.duration = j.end$. In addition, we have the parameters:

R_m - Capacity available on machine m, $\quad \forall m \in M$
$[a_j, b_j]$ - Time window for job j, $\quad \forall j \in J$
$time_{jl}$ - Duration of the l'th box for job j, $\quad \forall j \in J,\ \forall l \in L$
cap_{jl} - Capacity consumption of the l'th box for job j, $\quad \forall j \in J,\ \forall l \in L$

Let x_{jl} denote a binary variable which is 1 if box l is chosen for job j and 0 otherwise. Introducing the artificial job *makespan* with zero duration, we can state our model as follows:

$$
\begin{array}{lll}
\min & makespan.end & \\
\text{s.t.} & \sum_{l \in L} x_{jl} = 1 & \forall j \in J \quad (1) \\
& j.duration = \sum_{l \in L} (x_{jl} \cdot time_{jl}) & \forall j \in J \quad (2) \\
& j.capacity = \sum_{l \in L} (x_{jl} \cdot cap_{jl}) & \forall j \in J \quad (3) \\
& a_j \leq j.start & \forall j \in J \quad (4) \\
& j.end \leq b_j & \forall j \in J \quad (5) \\
\text{(IP/CP)} & j \ \ precedes \ \ makespan & \forall j \in J \quad (6) \\
& j \ \ precedes \ \ k & \forall (j,k) \in P \quad (7) \\
& cumulative \begin{pmatrix} \{j.start\}_{j \in M_m} \\ \{j.duration\}_{j \in M_m} \\ \{j.capacity\}_{j \in M_m} \\ R_m \end{pmatrix} & \forall m \in M \quad (8) \\
& x_{jl} \in \{0,1\} & \forall j \in J, \ \forall l \in L \quad (9)
\end{array}
$$

where *cumulative* is a *global constraint* in CP, stating that, at all times, the total capacity is not exceeded by the capacity consumption of running jobs. The constraint can be rewritten as

$$
\begin{array}{l}
cumulative\left((t_1,\ldots,t_n),(d_1,\ldots,d_n),(r_1,\ldots,r_n),R\right) \\
\Updownarrow \\
\sum_{\{j \mid t_j \leq t \leq t_j + d_j\}} r_j \leq R \quad \forall t
\end{array}
$$

where the vector $(t_1,\ldots,t_n)$ represents starting times of jobs $1,\ldots,n$, with duration $(d_1,\ldots,d_n)$ and capacity consumption $(r_1,\ldots,r_n)$. Available capacity is R.

The above constraints (1) choose a box for each job, yielding a specific time and capacity consumption in cooperation with (2) and (3). Constraints (4) and (5) consider time windows. Constraints (6) together with the objective function minimize the completion time of the last job. Constraints (7) handle precedence constraints (j precedes k means $j.end \leq k.start$), whereas (8) handles resource consumption for each machine.

3 Tabu search

To obtain a solution to the given problem we need the starting time and the box size for each job since then the completion times, the durations and the capacity consumptions are implicitly determined. However, the numerous possible starting times for each job prevent us from using the starting times explicitly in the solutions. Instead we use the box size for each job and a sequence which specifies the order of the starting times. In the following we let j_p denote the number of the job at position p in the corresponding sequence. Now the sequence specifies that since job j_1 is before job j_2 in the sequence, j_2 must start no earlier than j_1. A solution to a problem with 9 jobs is shown in Figure 3 where the sequence is defined by $j_1 \dots j_9$ and the box choices by the box numbers l_{j_p} stated below.

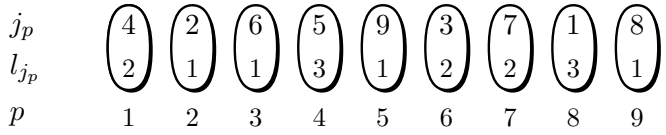

Figure 3. Sequence and box choices for example with 9 jobs.

Given both a box size for each job and a job sequence the corresponding optimal solution can be found or infeasibility can be proven. This means that the size of the solution space has been dramatically decreased without excluding optimal solutions. How to complete the solution to find the exact starting times for each job is discussed in Section 3.3.

Before the tabu search can start, an initial feasible solution is needed. A solution is feasible, if it is possible to schedule all jobs according to the sequence and the box sizes and still satisfy all time windows, capacity constraints and precedence constraints. It turns out that the problem of finding an initial solution is very hard, but a heuristic method for solving this problem is presented in Section 3.2.

Elements and features of the tabu search such as the neighbourhood, tabu lists, intensification strategies and diversification strategies are discussed in Sections 3.4, 3.5, 3.6 and 3.7, respectively. Part of the notation is inhereted from Chiang and Russell [13].

3.1 Preprocessing

In order to detect infeasible solutions quickly we tighten the time windows by considering precedence constraints. If a job j, has a time window $(0, t)$, but at the same time is a successor of another job $\hat{j}$, then the time window can be adjusted to start at the earliest completion time for job $\hat{j}$. To do this, a

precedence graph G is constructed where all jobs are represented by a node, and all precedence constraints by a directed arc between the two nodes involved, pointing away from the predecessor.

For all connected components in the graph the following procedure adjusts the beginning of the time windows. Let $C \subseteq G$ be a connected component, and let $j \in C$ be a job in C. Then a_j denotes the earliest starting time, and $time_{j1}$ denotes the minimal duration for job j. We let P_j denote all predecessors of job j, note $P_j \subset C$. The earliest starting times for the jobs in C are now adjusted by setting $a_j = \max\{a_j, a_i + time_{i1} \ \forall i \in P_j\}$ for all $j \in C$, but in an order such that all predecessors of j have been adjusted before j. Such an order exists, since otherwise a directed cycle would exist, and the jobs would be impossible to schedule. The latest completion times can be adjusted in a similar manner by starting with the jobs in C having no successors.

3.2 Initial solution

Garey and Johnson [14] have shown that, for a similar setup, the decision problem on determining the existence of a feasible schedule with a makespan less than a given deadline (in our case 8.00 am) is $\mathcal{NP}$-complete in the strong sense. In this section we shall describe a heuristical procedure to generate an initial feasible solution for this particular instance. The procedure is divided into five steps where the first three steps use the precedence graph to generate a sequence. In the fourth step, box sizes are chosen. If the solution obtained is feasible the procedure stops, and otherwise Step 5 relaxes the problem and uses the tabu search to find a feasible solution.

Step 1
Notice, to obtain a feasible solution, three groups of constraints must be fulfilled simultaneously, namely precedence constraints, time window constraints and capacity constraints. To ensure fulfilment of the precedence constraints, we use the precedence graph described in Section 3.1 to divide the jobs into *layers*. The successor of a job will always be in a higher layer than the job itself, and the jobs in one layer cannot start before all jobs in preceding layers have started. However, this dividing process faces the risk of assigning jobs with late time windows to an early processing layer. This could happen if an entire component of the precedence graph has to be processed after a certain time, but the first job is assigned to layer 1. Jobs from other components, which could be processed early, would then be stalled if they were in layer 2, and the entire schedule would be delayed. Hence in our derivation of layers, we introduce a variable *start* and initialize it to 0. Then we assign jobs that have no predecessors and are able to start before or at time *start*. All their successors, having no other predecessors and being able to start before or at

time *start*, are then scheduled in the next layer etc. When no more jobs can be assigned due to either time window constraints or precedence constraints, the variable *start* is increased by a constant amount of time, and a new level of layers can be derived with jobs being able to start before the new limit. The jobs are numbered consecutively, starting with the jobs on the lowest layer. This continues in an iterative fashion, until all jobs have been numbered, and we have a sequence containing all jobs.

Step 2
This step is very similar to Step 1 except the layers are generated backwards. This means that the layer containing the last jobs are generated first, and then the preceding layers are generated one by one. Again the successor of a job will always be in a higher layer than the job itself, and the job in one layer cannot start before all jobs in the preceding layer have started.

Step 3
The sequence from Step 1 has the disadvantage that all jobs without precedence constraints and time windows are scheduled first. This means that jobs which could have been scheduled later might delay some of the large components of the dependency graph. The sequence developed in Step 2 has the opposite problem since in this case the jobs with few constraints are scheduled very late and might cause jobs to break their time windows. Hence in this step we obtain a new sequence by taking a convex combination of the two sequences from Steps 1 and 2. This is done by calculating the convex combination of the positions in the two sequences for each job and then sequencing the jobs according to these numbers. Ties are broken arbitrarily. Notice that the new sequence still satisfies all precedence constraints.

Step 4
First we choose a box size for each job j on machine m according to the following scheme:

$$\begin{aligned} x_{j1} = 1 \quad & \text{if } 0 < cap_{j3} \leq \tfrac{R_m}{10} \\ x_{j2} = 1 \quad & \text{if } \tfrac{R_m}{10} < cap_{j3} \leq \tfrac{R_m}{4} \\ x_{j3} = 1 \quad & \text{if } \tfrac{R_m}{4} < cap_{j3} \end{aligned}$$

These choices have proven efficient in the particular problem. After the boxes have been chosen a check is made to see if the sequence obtained in Step 3 together with the box choices constitute a feasible solution.

Step 5
If the solution from Step 4 is infeasible we use the tabu search to find a feasible solution. The problem is relaxed by setting $b_j = \infty$ for all j, i.e. the time windows have no upper limit. Notice that this problem always has a feasible solution when the capacity requirement for each job is less than the capacity on the corresponding machine. The objective in this part of the tabu

search is to minimize the number of jobs which violate their time windows, and the search stops when a solution with value 0 has been found.

3.3 Completing a solution

As mentioned the solutions used in the tabu search only consist of a box choice for each job and a job sequence which determines the order of the starting times. This solution must be completed to include the exact starting and completion times for each job, since fulfilment of time windows and capacity constraints must be checked in order to prove feasibility of the solution. Since this check is done for all considered moves in each iteration the efficiency of the procedure has great influence of the overall performance of the tabu search.

Before the procedure is outlined it should be mentioned that the sequences given to the procedure always satisfy the precedence constraints, i.e. if j must be completed before $\hat{j}$ can start, then j will always precede $\hat{j}$ in the sequence.

The procedure exploits the fact that an optimal schedule with respect to the given sequence and box choices can be generated by scheduling one job at a time in the order of the sequence without backtracking. Since j_p is the job at position p in the sequence we know that when j_p is about to be scheduled all jobs $j_{\bar{p}}$ with $\bar{p} < p$ have been scheduled and $j_{p-1}.start \leq j_p.start$ due to the sequence. Furthermore, all the jobs that have been scheduled so far, start before or at $j_{p-1}.start$ and therefore the capacity consumption on each machine must be decreasing in time after $j_{p-1}.start$. The optimal starting time for j_p will hence be the first time after $\max\{a_{j_p},\ j_{p-1}.start\}$ and $\max\{j_{\bar{p}}.end | j_{\bar{p}}$ is predecessor of $j_p\}$ for which the capacity consumption, on the machine m used to process j_p, is less than or equal to $R_m - j_p.cap$. Hence a job is started the first time the four conditions shown in Figure 4 are fulfilled.

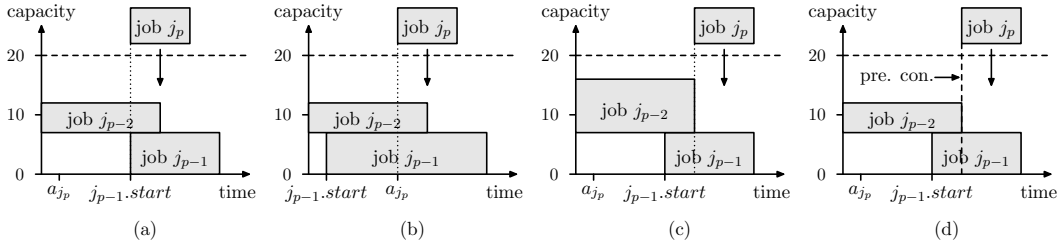

Figure 4. Scheduling the job j_p. The limiting constraints are: (a) The sequence, (b) time window, (c) capacity, (d) precedence constraint.

When the starting time of j_p has been determined the procedure checks if $j_p.end \leq b_{j_p}$ to see if the time window constraint is satisfied. If so, j_{p+1} is scheduled and otherwise the solution is infeasible and the procedure stops. If all jobs are scheduled we have a feasible solution since all constraints are satisfied, and the makespan is equal to $\max\{j.end | j \in J\}$.

3.4 Neighbourhood

To characterize the neighbourhood of a given solution $\bar{x}$ we define two kinds of moves. A *position move* (see Figure 5(a)) keeps the box sizes of $\bar{x}$ but changes the position of one job in the sequence, whereas a *box move* (see Figure 5(b)) maintains the job order of $\bar{x}$ but changes the box choice for a single job. Notice that, if the job j at position 5 in the job list is moved to position 2, not only does j get a new position, but the jobs at position 2, 3 and 4 are moved to the subsequent position. The neighbourhood for solution $\bar{x}$ can now be characterized as the union of solutions obtained by a single box move and solutions obtained by a single position move which fulfils the precedence constraints.

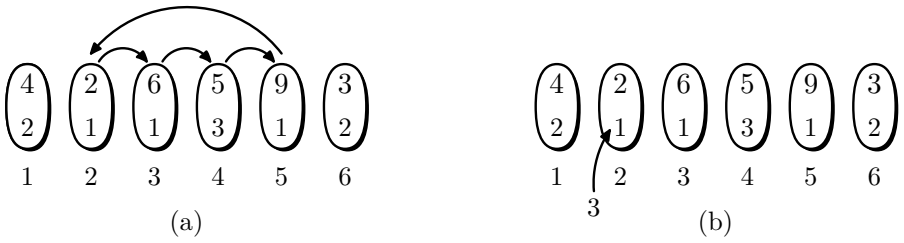

Figure 5. (a) position move, (b) box move.

The cardinality of the neighbourhood is $O(n^2)$ due to the large number of position moves, and in the present implementation we must consider approximately 120,000 moves (some are ignored due to violation of the precedence constraints) for each solution. The ability to select only part of the neighbourhood for examination is therefore crucial. We use two methods for limiting the number of possible moves.

3.4.1 Restricting position moves

By introducing a limit *movelimit* on how far a job can move, the number of considered position moves are reduced. This leads to faster iterations but might restrict the search from choosing some very good solutions. To avoid the search from stalling due to the restriction, the entire neighbourhood is examined every time the algorithm has performed non-improving moves for a predefined number of iterations. This makes the search capable of performing a single time consuming move and then a number of fast iterations to exploit the new conditions.

3.4.2 Candidate lists

The *Elite Candidate List* approach (see Glover and Laguna [7]), is used to limit the number of position moves by only evaluating moves belonging to

candidate lists. In this setup two lists are used, and they are constructed by evaluating the neighbourhood of the initial solution. All moves which lead to an improving makespan are stored in list 1, and all moves leading to the same makespan are stored in list 2. In the following iterations only moves from the two candidate lists are considered. First the moves in list 1 are evaluated, and if one of these moves leads to an improving makespan the best move is chosen. If list 1 does not contain an improving move the moves in list 2 are evaluated, and the best move considering both list 1 and list 2 is chosen. When a move has been chosen from one of the candidate lists both lists are updated by deleting all moves conflicting with the chosen one. This means that, if a position move for job j is chosen, then all other position moves for job j are deleted from the candidate lists and correspondingly for box moves. The candidate lists are used until no improving move has been found in the lists. When this happens both lists are deleted, and two new lists are generated by examining the possible moves of the current solution. Notice that, this might not be an evaluation of all possible moves, since the position moves might be restricted as explained in Section 3.4.1.

The underlying assumption of the strategy is that a move which performs well in the current solution will probably also lead to improvements in the following iterations.

3.5 Tabu list

The corner stone in tabu search is the use of short-term memory by generating a tabu list. The tabu list $TabuList$ stores the move from an iteration and keeps it for $TimeInTL$ iterations. This is done by keeping the iteration number $\hat{i}$, and when $\hat{i} + TimeInTL \leq iter$ the move is deleted from the list. The tabu list differentiates between the two kinds of moves, but the number of the job involved is always stored. If a box (position) move is performed for job j the tabu list restricts job j from performing a new box (position) move in the following $TimeInTL$ iterations, unless the *aspiration criterion* is satisfied. The aspiration criterion implemented checks if an improved makespan can be obtained by performing the forbidden move. If this is the case the tabu restriction is suspended, and the search is allowed to perform the move.

Storing all position or box moves for a single job in the tabu list is very restrictive, since it excludes a lot of moves, but it has been implemented due to the size of the solution space.

The tabu search implemented here has the ability to dynamically adjust the variable $TimeInTL$ which determines the number of iterations for which a move is tabu. $TimeInTL$ is decreased by the parameter $zdecrease = 0.9$

every time the search is trapped in a solution without a non-tabu or feasible neighbour and increased by $zincrease = 1.1$, when the same makespan has been found in many successive iterations.

In addition a variable *steps* is counting the number of moves without a change in $TimeInTL$, and $TimeInTL$ is decreased by *zdecrease* if *steps* exceeds a fixed threshold *movingaverage*. This adjustment helps the search to avoid a lot of bad moves which could be the result of a long tabu list.

3.6 Intensification strategy

The implementation of the intensification strategy is very similar to the implementation of the tabu list. A list $IntenArray$ holds moves which have led to improvements of the makespan. The moves are kept for $Intensize$ iterations, and corresponding moves for the same job are not allowed while the move is in the $IntenArray$. For example, if a position move is performed for job j in iteration $\hat{i}$, a new position move cannot be performed for j before iteration $\hat{i} + Intensize$. However, the intensification status is not considered if a job satisfies the aspiration criterion. In this case the job can be chosen even though the move is in the intensification array.

3.7 Diversification strategies

The algorithm contains two kinds of diversification strategies. The first strategy is active throughout the search and helps the algorithm to perform a thorough search in the current region of the solution space, while the other strategy forces the search to change the region.

3.7.1 Penalized move value

The quality of a move is measured by *movevalue*, which gives the difference between the current makespan and the makespan obtained by performing the move, $movevalue = newTime - curTime$. This *movevalue* could be used to guide the search, but in order to implement the first diversification strategy a *penalized move value pmv* is introduced. The *pmv* takes into account how many times the job has been moved before:

$$pmv = \begin{cases} movevalue + \alpha \cdot Move[j], & \text{if } movevalue \geq 0 \\ movevalue, & \text{if } movevalue < 0 \end{cases}$$

where $Move[j]$ counts the number of moves performed by job j and α is a parameter to adjust the penalty. By choosing moves according to lowest pmv, the algorithm automatically follows the diversification strategy.

3.7.2 Escape procedure

In order to move the search from one region of the solution space to another, an escape procedure is invoked when too many successive iterations have resulted in the same makespan. The procedure makes a number of random moves which lead the algorithm away from the current region. During the escape procedure only feasible moves are allowed, since a feasible solution must be available when all the moves are performed.

The general tabu search procedure adjusted according to the strategies above can be seen in Figure 6.

4 Computational results

Since the present problem is a large-scale scheduling problem, traditional IP/CP code is unable to solve it. We implemented the problem in OPL Studio (by ILOG [15]) and provided it with the search strategy to start with box choices according to the scheme in Step 4 of Section 3.2. OPL Studio with default setting was unable to solve the problem in 24 hours. In fact, within 24 hours OPL Studio was unable even to find a feasible solution to the problem, whereas our algorithm provided a feasible solution in 1 min, 5 sec. Notice that, all computational results reported in this section have been found using an Intel Xeon 2.67 GHz processor with 4 GB Ram.

By comparing the average makespan reported by Sonofon [4] and the makespan obtained by our algorithm with a naïve lower bound we show that significant improvements can be gained within a short amount of time.

To obtain the lower bound we disregard the precedence constraints. This allows us to schedule the three machines independently. Then for each job we let *total capacity consumption* be (capacity consumption $\times$ duration) for the smallest possible box (box 1). Now, for a particular machine we are able to construct a sequence by ordering the jobs according to the starting time of their time windows, ties are broken arbitrarily. When the jobs are scheduled according to this sequence and treated as totally elastic without variation of the total

[4] The average makespan was found using the historical data that constituted the specifications for the jobs.

```
1  procedure tabu search
2     time = 0
3     adjust time windows (3.1)
4     find initial solution (3.2)
5     iter = 0
6     while ((iter < maxiter)  ∧  (time < timelimit)) do
7        curmove = ∅
8        update TabuList
9        update IntenArray
10       if (candlist1 ∪ candlist2 = ∅) then
11          create new candlist with respect to restrictions (3.4.1 & 3.4.2)
12       end if
13       for all (moves in candlist1) do
14          complete the resulting solution (3.3)
15          check the TabuList and IntenArray (3.5 & 3.6)
16          if no improving move has been found check candlist2
17          choose curmove according to pmv (3.7.1)
18       end for all
19       if (curmove = ∅) then
20          decrease TimeInTL and let steps = 0 (3.5)
21       end if
22       else
23          update curSol by performing curmove
24          add the move to tabulist (3.5)
25          add the move to IntenArray if it leads to an improvement (3.6)
26       end else
27       update candlist (3.4.2)
28       if (iterations with same makespan = escaperepetion) then
29          use escapeprocedure (3.7.2)
30       end if
31       iter++
32    end while
33 end procedure
```

Figure 6. Pseudo code for the tabu search algorithm. Numbers in parentheses refer to the corresponding sections.

capacity we obtain a lower bound on the makespan. The lower bound for the present problem is 591 minutes.

The algorithm presented in this paper yields a makespan of 614 min, which is 3.89 percent above the lower bound. The average makespan obtained by Sonofon on the other hand is 821 min and hence is 38.92 percent above the lower bound. By a direct comparison of the two makespans it can be seen that our schedule saves 25.21 percent of scheduling time compared to the strategy implemented by Sonofon.

The best solution was found in 16 min, 44 sec, and hence the algorithm can be used on a daily basis to schedule the jobs which have to be processed during the night. Furthermore, Figure 7 shows that the significant improvements are obtained in a rather short amount of computation time, and afterwards only small improvements are made. This means that the algorithm is still applicable even though the job specifications are unknown until just prior to the actual scheduling process.

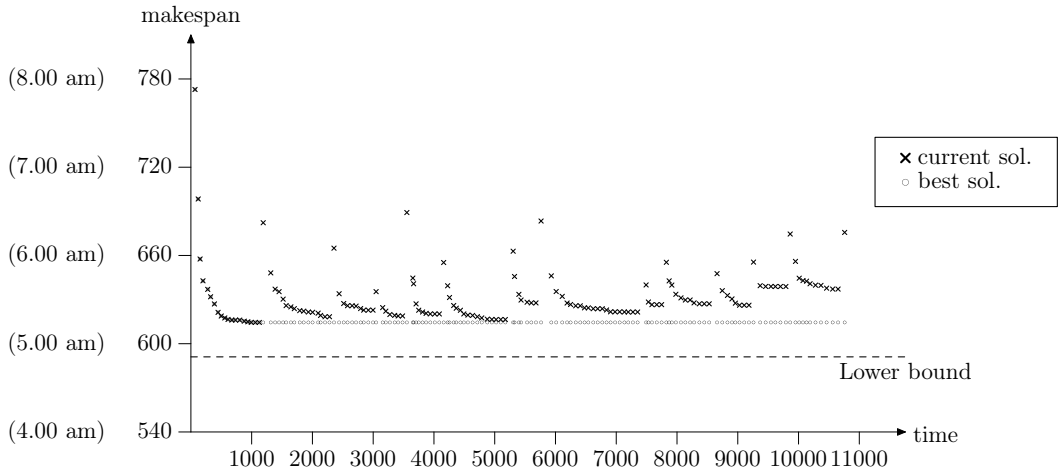

Figure 7. Makespan in minutes as a function of CPU time in seconds. A point is inserted after every 50 iterations of the algorithm.

In addition, the solution of the algorithm can be used to examine how the available capacity is used. Figure 8 shows a very uneven server exploitage during the night, and in particular if jobs were moved from machines 1 and 3 to machine 2 the makespan could be reduced.

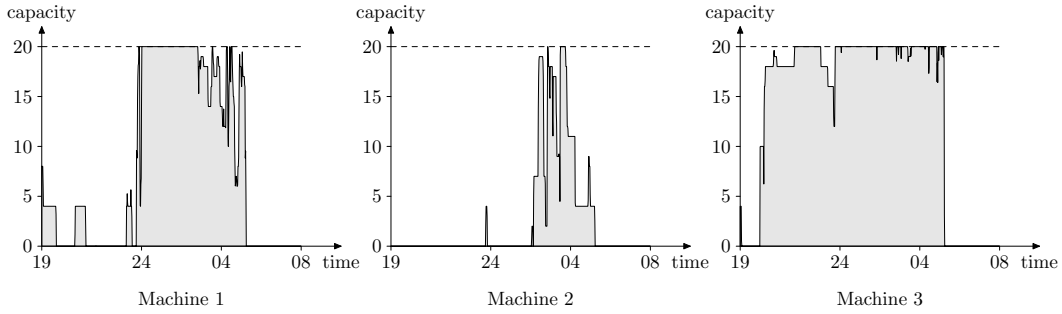

Figure 8. Capacity consumption on the three servers.

We have also tested the benefits of scheduling all jobs on one large server in stead of three separate ones. Our algorithm yields a makespan of 518 minutes in 1 hour, 4 minutes of computation time and therefore supports such an implementation. This scenario has been considered by Sonofon but is not implementable with their current hardware.

Acknowledgements

We would like to thank Morten Bech Kristensen (Sonofon), Lars Jørgensen (Sonofon) and Lars Grynderup (DM-Data) for data supply and many helpful discussions.

References

[1] Jain V, Grossmann IE. Algorithms for hybrid MILP, CP models for a class of optimization problems. INFORMS Journal on Computing 2001;13(4):258-276.

[2] Hooker JN, Ottosson G. Logic-based Benders decomposition. Mathematical Programming 2003;96:33-60.

[3] Hooker JN. Logic-based Benders methods for planning and scheduling. Lecture given at ISMP 2003.

[4] Baptiste P, Le Pape C. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. Constraints 2000;5(1-2):119-139.

[5] Baptiste P, Le Pape C, Nuijten W. Constraint-based scheduling: Applying constraint programming to scheduling problems. Norwell, Ma: Kluwer Academic Publishers, 2001.

[6] Glover F. Tabu search - Part I. ORSA Journal on Computing 1989;1(3):190-206.

[7] Glover F, Laguna M. Tabu search. Norwell, Ma: Kluwer Academic Publishers, 1997.

[8] Grabowski J, Wodecki M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers and Operations Research 2004;31(11):1891-1909.

[9] Ferland JA, Ichoua S, Lavoie A, Gagné E. Scheduling using tabu search methods with intensification and diversification. Computers and Operations Research 2001;28(11):1075-1092.

[10] Porto SCS, Kitajima JPFW, Ribeiro CC. Performance evaluation of a parallel tabu search task scheduling problem. Parallel Computing 2000;26:73-90.

[11] Chekuri C, Bender M. An efficient approximation algorithm for minimizing makespan on uniformly related machines. Journal of Algorithms 2001;41:212-224.

[12] Mansini R, Speranza MG, Tuza Z. Scheduling groups of tasks with precedence constraints on three dedicated processors. Discrete Applied Mathematics 2004;134:141-168.

[13] Chiang W-C, Russell RA. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. INFORMS Journal on Computing 1997;9(4):417-430.

[14] Garey MR, Johnson DS. Computers and intractability. A guide to the theory of NP-completeness. New York: W.H. Freeman and Company, 1979.

[15] ILOG, ILOG Optimization Suite - white paper 2001. URL: http://www.ilog.com.