ELSEVIER

European Journal of Operational Research 102 (1997) 528–537

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Heuristics for unrelated machine scheduling with precedence constraints

Jeffrey Herrmann [a], Jean-Marie Proth [a,b], Nathalie Sauer [b]

[a] *Institute for Systems Research, University of Maryland, College Park, MD 20742, USA*
[b] *INRIA-Lorraine, Technopôle Metz 2000, 4 rue Marconi, F-57070 Metz, France*

Received 20 February 1996; accepted 2 July 1996

Abstract

In this paper, we consider the problem of scheduling tasks on unrelated parallel machines. Precedence constraints between the tasks form chains of tasks. We propose a number of heuristics in order to find near optimal solutions to the problem. Empirical results show that the heuristics are able to find very good approximate solutions. © 1997 Elsevier Science B.V.

Keywords: Scheduling; Parallel resources; Makespan

1. Introduction

We consider the following problem. A set of workers must perform a finite set of tasks, and certain tasks cannot begin until others are completed. The time required to perform each task varies from worker to worker. We are interested in assigning the tasks to each worker and scheduling the tasks in such a way that the time needed to finish all of the work is minimized. In other words, we wish to minimize the maximal task completion time, also called the makespan.

Such a problem typically occurs in an office or project management environment, where the resources are workers who have different skills. The problem is called the Office Scheduling Problem. Due to the various skills available in an office the time necessary to complete a task can vary greatly from worker to worker. The tasks are the work that needs to be done during a specific period (a day for instance). Minimizing the makespan ensures that all of the tasks are done as soon as possible (ideally before the end of the assigned period). Furthermore, reducing the makespan leads to working periods which are not too different from worker to worker.

In general, the set of precedence constraints can be quite large. In this discussion, we assume that each task has at most one predecessor and at most one successor. Thus, the entire set of tasks includes chains of tasks. Completing a task enables the next task in the chain to begin.

This problem is an unrelated machine scheduling problem. Even without precedence constraints, minimizing the makespan is a *NP*-complete problem. However, a number of researchers have proposed and analyzed heuristics for this problem.

Potts (1985) uses a linear programming relaxation of the assignment problem formulation. This problem can be solved to assign most of the tasks to machines. However, the solution has up to $m - 1$ tasks divided between m machines. Given the partial solution from the *LP*, the heuristic uses an exponential time exhaustive search to determine the best assignment of these tasks. The worst case relative

PII S0377-2217(96)00247-0

performance of the heuristic is 2. If $m = 2$, an $O(n)$ heuristic is presented; this heuristic has worst case relative performance of 1.5.

Ibarra and Kim (1977) present a number of greedy heuristics similar to list scheduling. The worst case relative performance of these heuristics is m, but the heuristics are polynomial. If $m = 2$, the worst case relative performance is $(\sqrt{5} + 1)/2$.

Davis and Jaffe (1981) order on each machine all of the tasks by their efficiency, which is defined as the minimum processing time for the task divided by the processing time of this worker. Their heuristic schedules on the next available machine the most efficient task until the efficiency for all remaining tasks on the machine is too low. The effort of the heuristic is $O(mn\log n)$. The worst case relative performance is $2.5\sqrt{m}$.

Lenstra et al. (1990) use a sequence of linear programs to find the smallest deadline for which the linear programming relaxation has a feasible schedule. In the second phase, a bipartite matching problem is solved to assign the unscheduled jobs to machines. The worst case relative performance of this polynomial time heuristic is 2.

Hariri and Potts (1991) compare a number of heuristics for the problem. They consider two phase heuristics which start with an LP relaxation and then apply a matching or the earliest completion time heuristic (ECT) to assign the unscheduled jobs. They also consider the ECT procedure itself, proving that its worst case relative performance is $1 + {}^2\log n$. Finally, they consider reassignment and interchange heuristics to improve a schedule constructed by one of the above heuristics. They conclude that the use of improvement techniques with the ECT heuristic gives satisfactory solutions, though slightly better results can be achieved by using the two phase heuristics with the improvement techniques.

Van de Velde (1993) uses a surrogate relaxation problem to derive a lower bound. The search for the best lower bound is an ascent direction algorithm. An approximation algorithm based on the dual problem is presented and compared to solutions found with a branch-and-bound and other heuristics. A branch-and-bound over the job assignment is able to solve some quite large problems using less than 100 000 nodes.

In conclusion, it appears that the ECT heuristic with improvements provides simple but good approximation for the problem. If better solutions are required, the duality based algorithm would be preferred, since it does not require an enumeration or a sequence of linear programs.

The research on problems with precedence constraints has been restricted largely to identical parallel machines. Ullman (1975) proves that even if all the jobs have length one, the problem is NP-complete, although the problem can be solved in polynomial time if the precedence constraints form a tree (Hu, 1961). Graham (1966) considers the worst case performance of list scheduling in the presence of precedence constraints, which is $2 - 1/m$. Du et al. (1991) show that the two machine problem with arbitrary processing times and tree precedence constraints is strongly NP-complete. Cheng and Sin (1990) review a number of results on the worst case performance of list scheduling and highest-level-first approximation algorithms for minimizing makespan on identical parallel machines. Hoitmont et al. (1990) present a Lagrangian relaxation technique for the problem of minimizing total weighted quadratic tardiness on identical machines with precedence constraints. Outside of these approaches, however, few approximation algorithms have been proposed, and none consider nonidentical machines.

In this paper, we address the lack of heuristics for parallel, unrelated machine scheduling with precedence constraints. Throughout the remainder of the paper, we will use the term worker to describe a resource that can perform a task.

The problem is presented in Section 2. In Section 3, we propose a straightforward heuristic which assigns tasks to workers and schedules the tasks simultaneously. Section 4 is devoted to two heuristics that first assign tasks to workers, and then schedule the tasks, taking into account the precedence constraints. Two scheduling procedures are proposed. Section 5 presents lower bounds. In Section 6, we propose several numerical examples and evaluate the performance of the proposed algorithms. Section 7 concludes the paper.

2. Problem formulation

The problem can be formulated as follows. There exists a set of m workers W_j, $j = 1, \ldots, m$, and a

set E of n tasks T_i, $i = 1, \ldots, n$. Task T_i requires time p_{ij} when performed by W_j. We denote by $O(T_i)$ the task, if any, which can be released only if T_i is completed. In the rest of this paper, we refer to $O(T_i)$ as the immediate successor of T_i, which does not mean that $O(T_i)$ should start immediately when T_i is completed. $O^2(T_i) = O(O(T_i))$, and $O^q(T_i)$ is similarly defined for $q > 2$. Similarly, we denote by $O^{-1}(T_i)$ the task, if any, which is the immediate predecessor to T_i, i.e. which should be completed before the starting time of T_i. In a feasible schedule, each task must be performed by someone, and each worker performs at most one task at a time. The worker who begins a task finishes it without a break. T_i begins at time S_i and completes at time C_i. The goal is to find a feasible schedule that minimizes $C_{\max} = \max_{i \in \{1, \ldots, n\}} C_i$.

3. Heuristic $H1$

This heuristic assigns the tasks to the workers and defines the schedule simultaneously. The basic principle of this heuristic is very simple: it consists of scheduling at each iteration the task which could lead to a late schedule of some tasks in the future.

Let $k = 1$. E_k is the set of unscheduled tasks at iteration k. Let $E_1 = E$. F_k is the set of tasks that could be scheduled:

$$F_k = \{T_i \in E_k : O^{-1}(T_i) = \varnothing \vee O^{-1}(T_i) \notin E_k\}.$$

For each T_i in F_k, apply the following algorithm:

1. $S_i^o = C_k$ if $O^{-1}(T_i) = T_k$; $S_i^o = 0$ if $O^{-1}(T_i) = \varnothing$.
2. For $j = 1, \ldots, m$:
   2.1. $\mu_{ij} = \min\{t: t \geq S_i^o \wedge$ worker W_j is idle during $(t, t + p_{ij})\}$.
   2.2. $q = 1$. $S_i^q = \mu_{ij} + p_{ij}$
   2.3. If $O^q(T_i) = \varnothing$, go to 2.7. Else, define $r: T_r = O^q(T_i)$.
   2.4. For $h = 1, \ldots, m$:
   2.4.1. $\mu_{rh} = \min\{t: t \geq S_i^q \wedge$ worker W_h is idle during $(t, t + p_{rh})\}$.
   2.4.2. $\theta_{rh} = \mu_{rh} + p_{rh}$
   2.5. $S_i^{q+1} = \min_{h=1,\ldots,m} \theta_{rh}$.
   2.6. $q = q + 1$. Go to 2.3.
   2.7. $\theta_{ij} = S_i^q$.
3. Pick $j(i)$: $\theta_{ij(i)} = \min\{\theta_{ij}: j = 1, \ldots, m\}$.

Table 1
Processing times

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
W_1	3	4	8	2	5	9	3
W_2	9	5	2	6	10	4	8

Let T_{i^*} be the task to schedule next.

$$i^*: \theta_{i^*, j(i^*)} = \max_{T_i \in F_k} \theta_{i, j(i)},$$

$$S_{i^*} = \mu_{i^*, j(i^*)},$$

$$C_{i^*} = S_{i^*} + p_{i^*, j(i^*)},$$

$$E_k = E_k - \{T_{i^*}\}.$$

Schedule $W_{j(i^*)}$ to begin T_{i^*} at time S_{i^*}.

Repeat for $k = 2, \ldots, n$.

Let us consider the following example which concerns two workers and seven tasks.

Example. The processing times are given in Table 1.

Furthermore we have to consider the following partial order:

$$O(T_1) = T_3,\ O(T_3) = T_7,\ O(T_2) = T_6.$$

The solution to this problem is reached after seven iterations. The results obtained at each iteration are gathered in Table 2, where:

- F_k is the set of tasks in which the next task to be scheduled is selected,
- $\theta_{i^*, j(i^*)}$ is the earliest time when the last successor of T_{i^*} can be completed if T_{i^*} is assigned to $W_{j(i^*)}$,
- T_{i^*} is the selected task,

Table 2
Summary of iterations

k	F_k	$\theta_{i^*, j(i^*)}$	T_{i^*}	$W_{j(i^*)}$	S_{i^*}	C_{i^*}
1	$\{T_1, T_2, T_4, T_5\}$	8	T_1	W_1	0	3
2	$\{T_2, T_3, T_4, T_5\}$	9	T_2	W_2	0	5
3	$\{T_3, T_4, T_5, T_6\}$	10	T_3	W_2	5	7
4	$\{T_4, T_5, T_6, T_7\}$	11	T_6	W_2	7	11
5	$\{T_4, T_5, T_7\}$	10	T_7	W_1	7	10
6	$\{T_4, T_5\}$	15	T_5	W_1	10	15
7	$\{T_4\}$	5	T_4	W_1	3	5

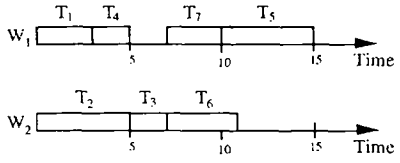

Fig. 1. Schedule when applying $H1$.

- $W_{j(i*)}$ is the processor which will perform T_{i*}.

4. Assignment heuristics

This section presents two heuristics that begin by assigning the tasks to workers while ignoring the precedence constraints. Because the problem has few precedence constraints (at most $n-1$), we expect that subsequently scheduling the tasks to satisfy these constraints will not significantly affect schedule quality. The first assignment heuristic applies a modified form on $H1$ (see Section 3) to schedule the tasks. The second assignment heuristic improves a feasible schedule by reducing the critical path. Both heuristics use simulated annealing to improve the final solution. Section 4.1 presents the assignment procedure where both heuristics begin. Section 4.2 discusses the two scheduling procedures. Section 4.3 presents the simulated annealing procedure.

4.1. Procedure A1: Assigning tasks to workers

Both assignment heuristics begin with Procedure $A1$, a branch-and-bound search that assigns tasks to workers while ignoring the precedence constraints. Procedure $A1$ is a breadth first search that constructs partial assignments. Each level of the search tree corresponds to a task, and each node represents a specific assignment. The procedure orders the tasks by their average processing times. That is, the task order is $T_{i1}, T_{i2}, \ldots, T_{in}$:

$$\sum_{j=1}^{m} p_{i_1 j} \geq \sum_{j=1}^{m} p_{i_2 j} \geq \cdots \geq \sum_{j=1}^{m} p_{i_n j}.$$

The corresponding tree is shown in Fig. 2. The optimal assignment $\{(T_i, W_{j(i)}): i = 1, \ldots, n\}$ minimizes the maximum total processing time for all workers.

At each node, Procedure $A1$ generates a lower bound and two upper bounds. Section 4.1.1 describes the lower bound. Section 4.1.2 discusses the two upper bounds.

Procedure $A1$ leads to the optimal assignment problem, that is to the minimal makespan when relaxing the precedence constraints. This algorithm is time consuming. To reduce the computation time, we developed procedure $A2$ which is the same as $A1$, except that it stops when $up_0 < \lambda lb_0$, where up_0 is the smallest upper bound, lb_0 is the smallest lower bound and λ is a real greater than 1.

Indeed, $A2$ does not lead to the optimal solution but to a good solution, and the computation time decreases drastically compared to the computation time of $A1$. As a consequence, the size of the problems which can be solved is much greater, as shown in Section 6.

4.1.1. The lower bound

Consider a node at level a. Tasks $T_{i1}, \ldots, T_{ia}$ have been assigned to workers $W_{j(i1)}, \ldots, W_{j(ia)}$. Let μ_j be the total processing time of the tasks assigned to W_j, $j = 1, \ldots, m$. Let E be the set of unassigned tasks. The lower bound is the solution to the following linear program:

min z:

$$z \geq \mu_j + \sum_{T_i \in E} x_{ij} p_{ij}, \quad \forall j = 1, \ldots, m, \tag{1}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall T_i \in E, \tag{2}$$

$$x_{ij} \geq 0, \quad \forall T_j \in E,\ j = 1, \ldots, m. \tag{3}$$

x_{ij} represents the fraction of T_i assigned to W_j. The right-hand side of constraint (1) measures the

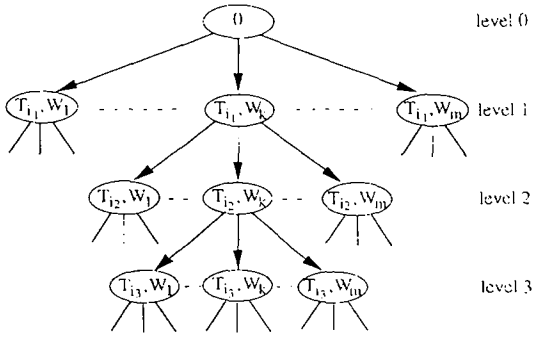

Fig. 2. The branch-and-bound tree.

total processing time of the tasks assigned to W_j. constraint (2) insures that the task is performed completely. z is the makespan. Note that constraint (3) relaxes the $0-1$ constraints of the assignment problem.

4.1.2. The upper bounds

Procedure $A1$ calculates two upper bounds at each node. First, $A1$ constructs an integer solution to the above linear relaxation. Second, it constructs an assignment using a modified form of heuristic $H1$. Both constructions are polynomial time algorithms.

4.1.2.1. Upper bound 1. Let x_{ij}^* be an optimal solution to the above linear program. In general, some x_{ij}^* will be between 0 and 1. For each $T_i \in E$, pick $j(i)$: $x_{i,j(i)}^* = \max\{x_{ij}^*: j = 1, \ldots, m\}$. Let $z_{ij} = 1$ if $j = j(i)$, 0 otherwise. The first upper bound is:

$$\max_{j=1,\ldots,m}\left\{\mu_j + \sum_{T_i \in E} z_{ij} p_{ij}\right\}.$$

4.1.2.2. Upper bound 2. Let $k = 1$. E_k is the set of unassigned tasks at the beginning of the kth iteration. $E_1 = E$. For each $T_i \in E_k$, apply the following algorithm:

1. For $j = 1, \ldots, m$, set $\theta_{ij} = \mu_j + p_{ij}$,
2. Pick $j(i)$: $\theta_{ij(i)} = \min\{\theta_{ij}: j = 1, \ldots, m\}$.

Pick i^*: $\theta_{i^*,j(i^*)} = \max\{\theta_{ij(i)}: T_i \in E\}$. Add $p_{i^*j(i^*)}$ to $\mu_{j(i^*)}$. $E_{k+1} = E_k \setminus \{T_{i^*}\}$. Add 1 to k and repeat until $E_k = \varnothing$. The second upper bound is $\max\{\mu_j: j = 1, \ldots, m\}$. This construction attempts to minimize the effect of large tasks by assigning them to the best workers first.

4.2. Procedures H2 and CP: Scheduling the tasks

The two assignment heuristics use different scheduling procedures. Procedure $H2$ applies a modified form of $H1$ to schedule the assigned tasks. Procedure CP randomly generates a feasible schedule and improves it by reducing the critical path.

4.2.1. Procedure H2

This scheduling procedure gives priority to the tasks which could delay future tasks. Let $W_{j(i)}$ be the worker that performs task T_i.

Let $k = 1$. $E_1 = E$. E_k is the set of unscheduled tasks at the beginning of iteration k. F_k is the set of tasks that could be scheduled:

$$F_k = \{T_i \in E_k: O^{-1}(T_i) = \varnothing \vee O^{-1}(T_i) \notin E_k\}.$$

For each T_i in F_k; apply the following algorithm:

1. $S_i^o = C_k$ if $O^{-1}(T_i) = T_k$; $S_i^o = 0$ if $O^{-1}(T_i) = \varnothing$,
2. $\mu_{ij(i)} = \min\{t: t \geq S_i^o, \wedge$ worker $W_{j(i)}$ is idle during $(t, t + p_{ij(i)})\}$,
3. $q = 1$. $S_i^q = \mu_{ij(i)} + p_{ij(i)}$,
4. If $O^q(T_i) = \varnothing$, go to 8. Else, define r: $T_r = O^q(T_i)$,
5. $\mu_{rj(r)} = \min\{t: t \geq S_i^q \wedge$ worker $W_{j(r)}$ is idle during $(t, t + p_{rj(r)})\}$,
6. $S_i^{q+1} = \mu_{rj(r)} + p_{rj(r)}$,
7. Add 1 to q. Go to 4,
8. $\theta_{ij(i)} = S_i^q$.

Let T_{i^*} be the task to schedule next.

$$i^*: \theta_{i^*,j(i^*)} = \max_{T_i \in F_k}\{\theta_{i,j(i)}\},$$

$$S_{i^*} = \mu_{i^*,j(i^*)},$$

$$C_{i^*} = S_{i^*} + p_{i^*,j(i^*)},$$

$$E_k = E_k - \{T_{i^*}\}.$$

Schedule $W_{j(i^*)}$ to begin T_{i^*} at time S_{i^*}.
Repeat for $k = 2, \ldots, n$.

4.2.2. Procedure CP

Procedure CP randomly generates a feasible schedule that respects the assignments made by procedure $A1$ and then repeatedly improves the schedule by reducing the critical path. At each iteration, the procedure must calculate the slack for each task and interchange two tasks.

4.2.2.1. Step 1: Generating a feasible schedule. Let $k = 1$. $E_1 = E$. E_k is the set of unscheduled tasks at the beginning of iteration k. F_k is the set of tasks that could be scheduled:

$$F_k = \{T_i \in E_k: O^{-1}(T_i) = \varnothing \vee O^{-1}(T_i) \notin E_k\}.$$

Select at random some T_i in F_k; apply the following algorithm:

1. $S_i^o = C_k$ if $O^{-1}(T_i) = T_k$; $S_i^o = 0$ if $O^{-1}(T_i) = \varnothing$,
2. $\mu_{ij(i)} = \min\{t: t \geq S_i^o \wedge$ worker $W_{j(i)}$ is idle during $(t, t + p_{ij(i)})\}$,

$S_i = \mu_{ij(i)}$; $C_i = S_i + p_{ij(i)}$; $E_{k+1} = E_k \setminus \{T_i\}$.

Schedule $W_{j(i)}$ to begin T_i at time S_i. Repeat for $k = 2, \ldots, n$.

4.2.2.2. Step 2: Compute the slack for each task. Define i^+ as the task T_i^+ that worker $W_{j(i)}$ performs after T_i. If T_i is the last task, T_i+ does not exist. $T_{i(1)} = O(T_i)$ if the successor exists. Let $D(i)$ be the slack of T_i. Delaying T_i by an amount $d \le D(i)$ does not increase the schedule makespan.

$$D(i) = \min\{C_{\max} - C_i, D(i(1)) + S_{i(1)} - C_i, D(i^+) + S_{i^+} - C_i\}.$$

The second term vanishes if $O(T_i)$ does not exist. The third term vanishes if T_i^+ does not exist.

The critical path is a sequence of s tasks $T_{i1}, \ldots, T_{is}$ such that $S_{i1} = 0$, $S_{ik} = C_{ik-1}$ $(k = 2, \ldots, s)$, and $C_{is} = C_{\max}$. To reduce the makespan of the schedule, it is necessary (though not sufficient) to reduce the length of the critical path.

4.2.2.3. Step 3: Interchange two tasks. Select a critical path for the schedule and consider this sequence of tasks. Unless one worker performs the entire sequence, construct a set F of critical tasks using the below algorithm. Performing a task in F earlier may improve the schedule.

For each critical task $T_i = T_{ik}$ $(k > 1)$ perform the following steps; when the algorithm stops, repeat for the next critical task.

1. Verify that the same worker also performs the previous task $T_h = T_{ik-1}$. If $j(h) \neq j(i)$, stop. Else, let $j = j(i)$.
2. Verify that the previous task is not its predecessor. If $O^{-1}(T_i) = \varnothing$, let $S_i^o = 0$; go to 3. If $T_h = O^{-1}(T_i)$, stop. Else, let $S_i^o = C_p$ where $T_p = O^{-1}(T_i)$.
3. Determine the time that the worker is available: Let T_r be the task that W_j performs before T_h $(T_r^+ = T_h)$. If no such T_r exists, let $\mu_j = 0$; else $\mu_j = C_r$.
4. Determine the time that the previous task is available. If $O^{-1}(T_h) = \varnothing$, let $S_h^o = 0$. Else, let $S_h^o = C_q$ where $T_q = O^{-1}(T_h)$.
5. Calculate the earliest completion time of T_i after interchanging T_i and T_h: $\theta_i = \max(S_i^o, \mu_j) + p_{ij}$.
6. Calculate the earliest completion time of T_h after interchanging T_i and T_h: $\theta_h = \max(S_h^o, \theta_i) + p_{hj}$.
7. Verify that this interchange will not delay the next task that worker W_j performs. If T_i^+ exists and $\theta_h > S_{(i^+)} + D_{(i^+)}$, stop.
8. Verify that this interchange will not delay the previous task's successor. If $T_s = O(T_h)$ exists and $\theta_h > S_s + D(s)$, stop.
9. Let $\Delta_i = \min\{S_{(i^+)} + D_{(i^+)}, S_s + D(s)\} - \theta_h$ and add T_i to F. If T_i^+ does not exist, the first term vanishes. If T_s does not exist $(O(T_h) = \varnothing)$, the second term vanishes.

If $F = \varnothing$, then we cannot improve this critical path. Procedure *CP* stops. If more than one such task exists, select $i*$: $\Delta_{i*} = \max\{\Delta_i : T_i \in F\}$. Interchange T_{i*} and the task T_h that immediately precedes it. Recalculate the completion times for all tasks and return to Step 2.

4.3. Procedure SA: Refining the final solution

Both assignment heuristics use this simulated annealing procedure to improve the solutions that heuristics *H2* and *CP* construct. The procedure begins with the heuristic solution and searches for better schedules by iteratively constructing new solutions that are similar to the current solution.

Initially, the current schedule is the one that the scheduling heuristic generates. If the current schedule $S1$ has makespan z_1, Procedure *SA* performs the following steps to find a neighbor $S2$:

1. Randomly choose a worker W_j.
2. Randomly choose two tasks T_i and T_r that W_j performs.
3. Let $T_{i1}, \ldots, T_{is}$ be the sequence of tasks that W_j performs: $T_{i1} = T_i$ and $T_{is} = T_r$.
4. Determine if any task is a successor to T_i: If there exist $1 < k \le s$ and $q \ge 1$: $T_{ik} = O^q(T_i)$, then return to 1.
5. Determine if any task is a predecessor to T_r: If there exist $1 \le k < s$ and $q \ge 1$: $T_r = O^q(T_{ik})$, then return to 1.
6. Interchange T_i and T_r and construct the corresponding feasible schedule $S2$. Let z_2 be the makespan.
7. If $z_2 \le z_1$, unconditionally accept $S2$ as the current schedule. Otherwise; accept $S2$ with probability p, where $\ln p = -(z_2 - z_1)/T$ and T is the current annealing temperature.

The procedure continues in this manner, constructing a neighbor of the current solution by interchanging two tasks, accepting a better solution, and occasionally accepting a slightly worse solution to diversify the search. The initial temperature is $T_0 = 100$. After *SA* accepts 50 solutions at temperature T_k or rejects 50 solutions at temperature T_k, the temperature is reduced to $T_{k+1} = 0.98T_k$. Procedure *SA* stops when $T_{k+1} < 0.01$ $(k = 457)$ or when it constructs a solution that achieves a lower bound.

5. Lower bounds

We use two criteria to evaluate the heuristics: computation time and solution quality. Because we cannot find (in reasonable time) the optimal solutions, we compare the approximate solutions that the heuristic generates to lower bounds on the optimal objective function value. We use three different lower bounds: *LB*1, *LB*2, and *LB*3. *LB*3, however, requires the task assignments that Procedure *A*1 generates.

5.1. Lower bound LB1

This lower bound is given by:

$$LB1 = \max_{i/T_i \in F} \left\{ \sum_{q=0}^{N_i} \min_{j \in \{1,\ldots,m\}} p_{i(q),j} \right\}, \tag{4}$$

where $F = \{T_i \in E / O^{-1}(T_i) = \varnothing\}$, the set of tasks with no predecessors, $i(0) = i$, $i(q)$ is such that $T_{i(q)} = O^q(T_i)$ if $q \geq 1$, and N_i is the number of successors of $T_i \in F$.

*LB*1 is a good lower bound if one chain of tasks dominates the other chains.

5.2. Lower bound LB2

This lower bound is given by:

$$LB2 = \frac{\sum_{i=0}^{n} \min_{j \in \{1,\ldots,m\}} p_{i,j}}{m}. \tag{5}$$

*LB*2 distributes the sum of the minimal processing times among the workers. Note that this lower bound is never greater than the makespan obtained by applying Procedure *A*1. But *LB*2 is useful when

Table 3
Small-sized numerical examples ($n \leq 30$)

Example #		1	2	3	4	5	6	7	8	9	10	11
n		14	28	16	17	27	16	12	11	19	15	29
m		8	7	8	4	4	5	6	4	6	6	4
NC		5	8	6	7	1	7	2	3	7	7	7
*H*1	*M*	23	37	22	54	92	32	26	40	**43**	**34**	88
	CT	1″	3″	1″	1″	2″	0″	0″	0″	1″	1″	2″
*A*1	*MW*	16	23	12	43	53	23	15	32	28	19	63
	CT	17″	1′16″	9″	30″	1′42″	16″	7″	2″	1′43″	6″	18″
*A*1, *H*2	*M*	30	27	20	56	54	**25**	**19**	40	51	42	73
	CT	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″
*A*1, *H*2	*M*	30	**23**	**17**	54	**53**	**25**	**19**	40	51	42	**63**
SA	*CT*	0″	0″	1″	21″	0″	0″	0″	0″	0″	0″	1″
*A*1, *CP*	*M*	30	**23**	**17**	54	**53**	26	**19**	44	51	42	**63**
	CT	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″
*A*1, *CP*	*M*	30	**23**	**17**	54	**53**	**25**	**19**	40	51	42	**63**
SA	*CT*	0″	0″	0″	21″	0″	1″	0″	0″	0″	0″	0″
*LB*1		16	17	17	40	22	25	19	32	43	34	30
*LB*2		9.25	20	9.25	38.25	49	19.4	10.33	26	22.67	15.5	57.75
*LB*3		30	19	17	43	22	25	19	40	51	42	34

Table 4
Medium-sized numerical examples ($n \leq 50$)

Example #		12	13	14	15	16	17	18	19	20	21	22
n		35	28	32	35	37	27	29	36	50	44	26
m		6	9	6	10	8	7	10	6	9	9	7
NC		10	6	2	8	7	11	13	14	16	15	6
$H1$	M	51	23	48	27	45	36	**30**	57	43	45	31
	CT	4″	4″	4″	6″	5″	2″	4″	4″	10″	7″	2″
$A1$	MW	36	16	30	18	26	26	14	35	28	26	21
	CT	6′32″	18′05″	3′10″	2h52′20″	32′02″	1h04′22″	47′54″	3′12″	2h25′44″	1h01′41″	2′03″
$A1$, $H2$	M	46	17	**30**	20	28	34	40	40	33	32	28
	CT	0″	0″	0″	0″	0″	0″	0″	1″	0″	0″	0″
$A1$, $H2$	M	**36**	17	**30**	**18**	**26**	34	40	**35**	**28**	27	**21**
SA	CT	10″	23″	0″	25″	22″	0″	0″	25″	30″	27″	22″
$A1$, CP	M	38	24	**30**	25	29	34	40	36	38	30	22
	CT	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″
$A1$, CP	M	**36**	17	**30**	**18**	**26**	34	40	**35**	**28**	28	**21**
SA	CT	7″	23″	0″	11″	7″	0″	0″	7″	15″	26″	6″
$LB1$		21	12	14	16	18	33	30	22	24	18	17
$LB2$		32.5	12.44	27	14.2	23.75	22.43	11	31.33	25.11	23.89	18.43
$LB3$		23	17	14	18	19	34	40	30	24	18	17

the size of the problem is too large to apply Procedure $A1$.

5.3. Lower bound LB3

We use this lower bound to evaluate the performance of Procedures $H2$ and CP, given the assignments generated by Procedure $A1$. Recall that T_i is assigned to $W_{j(i)}$. $LB3$ is given by:

$$LB3 = \max_{T_i \in F} \left\{ \sum_{q=0}^{N_i} p_{i(q),j(i(q))} \right\}. \tag{6}$$

Table 5
Medium-sized numerical examples with procedure $A2$

Example #		12	13	14	15	16	17	18	19	20	21	22
$A2$	MW	38	17	31	20	28	28	15	37	29	30	24
	CT	2″	22″	2″	12′52″	4″	2″	19′05″	2″	5″	5″	5″
$A2$, $H2$	M	41	22	31	24	29	43	38	38	33	36	28
	CT	0″	0″	0″	0″	0″	0″	0″	0″	1″	0″	1″
$A2$, $H2$	M	38	17	31	20	28	40	38	37	29	30	24
SA	CT	6″	0″	0″	12″	1″	1″	0″	9″	13″	11″	0″
$A2$, CP	M	38	24	31	27	28	40	38	37	38	34	24
	CT	0″	0″	0″	0″	0″	0″	0″	0″	0″	1″	0″
$A2$, CP	M	38	17	31	20	28	40	38	37	29	30	24
SA	CT	0″	0″	0″	9″	0″	0″	0″	0″	13″	11″	0″
$LB3$		26	17	16	19	19	40	38	28	24	18	21

6. Empirical results

Several numerical examples are given in the following tables. *NC* is the number of precedence constraints, *SA* denotes the simulated annealing, *M* is the makespan, *MW* is the makespan obtained by relaxing the precedence constraints, and *CT* is the computation time. The other notations are those used in the previous sections. Numerical results are presented in Tables 3–6.

The values in bold characters are optimal makespans. We know that a value obtained using an heuristic algorithm is optimal when it is equal to the greatest lower bound, i.e. when it is equal to max($a1$, $LB1$, $LB2$), where $a1$ is the value provided by algorithm $A1$.

As we can see, heuristic $H1$ is very fast and sometimes provides the optimal makespan. Furthermore, $H1$ can be applied whatever the size of the problem. Unfortunately, in some circumstances, the solution provided by $H1$ is far away from the optimal one.

The most time consuming algorithm is the branch-and-bound algorithm $A1$, the goal of which is to reduce the makespan after relaxing the precedence constraints. This algorithm is completed by $H2$ or by CP. The computation times given in the rows related to $H2$ and CP in Tables 3–6 concern only $H2$ and CP. These times are negligible.

In Tables 3 and 4, both algorithms ($A1 + H2$) and ($A1 + CP$) are refined using simulated annealing. This refinement step stops either when the criterion reaches max ($a1$, $LB1$, $LB2$, $LB3$), or when the temperature of the simulated annealing becomes less than a threshold given by the user. The computation times provided in the rows related to *SA* concern the simulated annealing only.

In Table 5, we provide the solutions of the same examples as the ones presented in Table 4. The only difference is that we replaced procedure $A1$ by procedure $A2$ with $\lambda = 1.2$ (see Section 4.1).

As we can see, when using $A1$, the computation times are much better, and the values of the criterion are close, and sometimes better, than those obtained in Table 4. Furthermore, these results are, in most of the cases, much better than those obtained using procedure $H1$.

Finally, we present in Table 6 much larger exam-

Table 6
Large-sized numerical examples (n > 50)

Example #		23	24	25	26	27	28	29	30	31	32	33
n		66	65	57	74	54	74	51	59	58	68	57
m		16	14	12	18	16	19	17	15	13	12	13
NC		18	4	16	18	17	10	24	19	22	4	15
H_1	M	23	30	30	22	20	22	**21**	26	27	37	28
	CT	11″	11″	7″	16″	7″	18″	6″	9″	7″	10″	7″
A_2	MW	17	20	21	18	16	14	12	18	20	23	19
	CT	23″	5″04″	10″	2″06″	1″30″	32″42″	59″	17″	13″	16″	13″
A_2, H_2	M	21	20	26	20	24	16	**21**	20	22	23	21
	CT	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″	0″
A_2, H_2	M	18	20	**21**	18	20	14	**21**	19	21	23	19
SA	CT	38″	0″	18″	18″	37″	20″	0″	19″	42″	0″	13″
A_2, CP	M	19	20	23	20	20	16	**21**	22	26	23	21
	CT	0″	0″	0″	0″	0″	1″	0″	0″	0″	0″	0″
A_2, CP	M	18	20	**21**	18	20	14	**21**	19	22	23	19
SA	CT	39″	0″	17″	21″	37″	16″	0″	29″	41″	0″	18″
$LB1$		11	12	21	17	15	6	21	17	20	12	14
$LB2$		13.81	15.5	17.25	13.94	10.81	11.21	9.71	14.53	17	19.25	15.92
$LB3$		11	15	21	17	18	13	21	19	20	15	14

ples performed using procedure $A2$. Most of the tests lead to a good, and sometimes optimal, solution in a reasonable amount of time.

7. Conclusions

The office scheduling problem requires one to perform chains of tasks with workers that have different skills. To solve this problem we developed an intuitive heuristic that constructs a feasible schedule directly and two assignment heuristics that first assign the tasks to workers and then schedule the tasks to satisfy the precedence constraints. Because the number of precedence constraints is limited, this second step does not significantly degrade schedule performance. Finally, we use a simulated annealing procedure to improve the final solution. Although the office scheduling problem is a computationally difficult question, these heuristics are able to construct high quality solutions in reasonable time.

References

UNLINKED

Cheng, T.C.E., and Sin, C.C.S. (1990), "A state-of-the-art review of parallel machine scheduling research", *European Journal of Operational Research* 47, 271–292.

Darema, F., Kirkpatrick, S., and Norton, V.A. (1987), "Parallel algorithms for chip placement by simulated annealing", *IBM Journal of Research and Development*, 31(3), 391–402.

Davis, E., and Jaffe, J.M. (1981), "Algorithm for scheduling tasks on unrelated processors", *Journal of the Association for Computing Machinery*, 28(4), 721–736.

Du, J., Leung, J.Y.-T., and Young, G.H. (1991), "Scheduling chain structured tasks to minimize makespan and mean flow time", *Information and Computation* 92, 219–236.

Graham, R.L. (1966), "Bound on certain multiprocessing anomalies", *Bell System Technical Journal* 45, 1563–1581.

Hariri, A.M.A., and Potts, C.N. (1991), "Heuristics for scheduling unrelated parallel machines", *Computers and Operations Research* 18(3), 323–331.

Hoitmont, D.J., Luh, P.B., Max, E., and Pattipati, K.R. (1990), "Scheduling jobs with simple precedence constraints on parallel machines", *IEEE Control Systems Magazine*, 10(2), 34–40.

Ibarra, O.H., and Kim, C.E. (1977), "Heuristic algorithms for scheduling independent tasks on nonidentical processors", *Journal of the Association for Computing Machinery*, 24(2), 280–289.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C. (1989), "Optimization by simulated annealing: An experimental evaluation; Part 1: Graph partitioning", *Operations Research*, 37(6), 865–892.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983), "Optimization by simulated annealing", *Science*, 220, 4598.

Lenstra, J.K., Shmoys, D.B., and Tardos, E. (1990), "Approximation algorithms for scheduling unrelated parallel machines", *Mathematical Programming*, 46, 259–271.

Potts, C.N., 1985. "Analysis of a linear programming heuristic for scheduling unrelated parallel machines", *Discrete Applied Mathematics*, 10, 155–164.

Ullman, J.D., "NP-complete scheduling problems", *Journal of Computing Systems Science*, 10, 384–395.

Van de Velde, S.L., "Duality based algorithms for scheduling unrelated parallel machines", *ORSA Journal on Computing*, 5(2), 192–205.