ELSEVIER

Discrete Applied Mathematics 65 (1996) 285–318

DISCRETE APPLIED MATHEMATICS

On a scheduling problem in a robotized analytical system

Alain Hertz*, Yves Mottet, Yves Rochat

Département de Mathématiques, Ecole Polytechnique Fédérale de Lausanne, Chaire de Recherche Opérationnelle, CH-1015 Lausanne, Switzerland

Received 15 February 1993; revised 1 April 1994

Abstract

We consider a robotized analytical system in which a chemical treatment has to be performed on a set of n identical samples. The same predetermined ordered set of tasks has to be performed on each sample. A unique robot has to transport the samples between the resources. The time spent by a sample in a resource is bounded by given minimal and maximal values. No-wait constraints and additional requirements such as blocking resources or limited capacities have to be taken into account. The objective is to perform the chemical treatment on the whole set of n samples in the shortest possible time, while satisfying all the constraints.

We describe a fast heuristic scheduling algorithm for solving this problem and apply it to a robotized sample preparation of membrane fatty acid esters for the identification of bacteria.

The proposed algorithm was implemented in practice and has proven to be satisfactory.

Keywords: Scheduling problems; No-wait constraints; Robotized system; Heuristic methods

1. Introduction

This article considers the problem of scheduling the chemical treatment of n identical samples in a robotized analytical system. It is motivated by a real life application: the robotized sample preparation of membrane fatty acid esters for the identification of bacteria.

The same predetermined ordered set of tasks has to be performed on each sample. Each task can be viewed as an amount of time (not necessarily fixed in advance) that a sample should spend in a resource of a given type. The samples may have to visit several times the same resource. The resources (e.g. a rotator or a bath) may have a capacity larger than one. This means that more than one sample can be treated simultaneously in a same resource.

* Corresponding author.

SSDI 0166-218X(95)00039-9

There is a unique robot in the laboratory. It has to place the samples into the resources, remove them from the resources and transport them between the resources. The robot can handle only one sample at a time.

For some pairs (i, j) of tasks which have to be performed on each sample, it is imposed that the time between the end of i and the beginning of j (task j is not necessarily an immediate successor of i) cannot be larger than a given maximal value. For security reasons, no violation of such a constraint is allowed since unexpected chemical reactions could occur.

Additional requirements such as blocking resources have to be taken into account. The chemical treatment should be performed in the shortest possible time.

A number of survey papers and books have dealt with deterministic scheduling problems (e.g. [1,4–6,9,12,22–25]). Nowadays, the interest in no-wait scheduling problems is increasing (e.g. [10,11,15,20,29–33]): these problems occur in a production environment in which a job (a sample) must be processed from its start to its finish, without any interruption either on or between machines (resources). In the problem that we consider, interruptions between machines are allowed but their duration is bounded by given maximal values. Longer interruptions would either inhibit chemical reactions or change characteristics of the sample such as the temperature or the viscosity.

Flowshop problems with a robot or an AGV taking care of the transportation of the jobs between the machines have been studied by several authors. In the case of a two-machine robotized system producing a single part type, Sethi et al. [34] have described an optimal policy that maximizes the long-run average production rate over the class of all policies. For a three-machine robotic cell and a single part type, they have determined an optimal policy over the class of cyclic policies where each cycle produces only one part. The later problem has also been investigated by Hall et al. [15]. For many part types and two machines, Sethi et al. [34] show that the problem can be formulated as a solvable case of the traveling salesman problem. This problem has also been considered in [13,19,26,27].

An extensive review of the literature on the scheduling of parts and robot moves in robotized systems is given in [34]. More recent studies can be found in [13–15,17]. It appears that our problem belongs to a class of scheduling problems which has not been treated in the classical literature. The most important differences are the following:

- a part (or sample) may have to visit several times the same resource;
- the duration of a task (i.e. its processing time) is not fixed in advance but bounded by given minimal and maximal values;
- the capacity of the resources may be larger than one. Hence, more than one sample can be treated simultaneously in the same resource;
- the waiting time between the end of a task and the beginning of another task may be bounded.

In the next section, we describe the scheduling problem in more detail. A heuristic scheduling algorithm is presented and evaluated in Sections 3 and 4. Extensions are discussed in the last section.

For potential users of the proposed algorithm, it is relevant to mention that it was actually implemented in practice and has proven to be satisfactory.

2. Description of the problem

In this section we describe the scheduling problem in more detail. Each resource r of the robotized analytical system has a limited capacity c_r: this means that at most c_r samples may be treated in r at the same time. The activation of a resource may be *explicit* or *implicit*. An implicit activation means that the chemical process begins as soon as the sample enters the resource. This is the case, for example, when a sample is introduced into a bath. In resources with explicit activation, the chemical treatment starts only once the resource has been activated manually or by a computer. For example, this may occur in a rotator. For resources with explicit activation and capacity larger than one, it is assumed that each position in the resource can be activated or desactivated independently.

In addition to these resources, there is a robot whose task consists of transporting the samples between the resources. The robot can handle at most one sample at a time.

A resource is said to be *blocking* if, once a sample has been introduced into it, the robot has to wait until the end of the task for moving the sample to the next resource. This may occur, for example, when a test tube must be fed with some reagent. In such a case, the robot places the test tube in a dispenser and unscrews the cap which remains in its grip. It cannot treat any other sample before the cap has been screwed back on the tube.

The same chemical treatment has to be processed on n identical samples: it consists of an ordered set $\{0, 1, \ldots, p\}$ of tasks. At time zero, all samples are staying in an initial resource $r(0)$ (e.g. a refrigerator). The robot is located at resource $r(0)$ and is ready for transporting any sample from $r(0)$ to another resource. The stay of the samples in $r(0)$ is considered as task 0 and may eventually be limited in time. The last task p (which is not limited in time) is the stay of the samples in a final resource $r(p)$. The capacities $c_{r(0)}$ and $c_{r(p)}$ are supposed to be larger or equal to n.

Each task t $(1 \leqslant t \leqslant p-1)$ can be viewed as an amount of time that a sample has to spend in a given resource $r(t)$. The part of this time during which $r(t)$ is active will be called the *active period* of t. The active period of a task cannot be interrupted and its duration is bounded by given minimal and maximal values.

There are two ways for dealing with a sample α for which the duration of the active period of the task currently performed on it has reached its maximal value. If the task is performed in a resource with an explicit activation then it can be desactivated: the sample α will then simply wait in the resource until the robot moves it to a next resource. In the case of an implicit activation, the robot has to remove α from the resource immediately. It may happen that the next resource is not available (for capacity reasons). In such a case, the robot has to place sample α into a storage

resource (if any). In this paper, we consider the case where there are no storage resources. This means that, if a sample should immediately be removed from a resource $r(t)$, then the number of samples currently in $r(t+1)$ is strictly smaller than $c_{r(t+1)}$.

For some pairs (i, j) of tasks, the time between the end of i on a sample and the beginning of j on the same sample cannot be larger than a given maximal value. Notice that j is not necessarily an immediate successor of i.

The objective is to perform the chemical treatment on the whole set of n samples in the shortest possible time. All constraints have to be satisfied since a modification of the chemical process could create unexpected reactions.

As already mentioned in the introduction, this problem belongs to a class of scheduling problems which has not been treated in the classical literature. Complexity results as those contained in [14, 15, 34] do not cover our problem.

The flowshop problem in a robotic cell, with a single part type, is a special case of our scheduling problem where it is assumed that all resource capacities are equal to one, the duration of each task is fixed and no part (sample) has to visit several times the same resource. The complexity of this problem is open, even if it is assumed that there are only three resources. Sethi et al. [34] have shown that an optimal one-part cycle (i.e. a cycle that produces a single part) can be determined in polynomial time. It has however never been proved that the repetition of one-part cycles dominates all other policies.

As pointed out by Goyal and Sriskandarajah [11], no-wait scheduling problems except for a few special cases belong to the class of NP-complete problems. On the one hand, our problem is more general than no-wait scheduling problems since no-wait constraints appear as a special case where it is imposed for each pair $(i, i+1)$ of consecutive tasks that task $i+1$ should immediately follow task i. On the other hand, we have to solve a scheduling problem with a single part (sample) type since the same ordered set of tasks has to be performed on each sample. It follows that the input size of the problem does not depend on the number n of samples.

Consider the simple problem represented in Fig. 1. Each sample α is initially staying in resource $r(0)$ and has to be moved to resource $r(1)$ where task 1 will be performed on it. Resource $r(1)$ has an implicit activation. The duration d of task 1 must satisfy $\underline{d} \leqslant d \leqslant \bar{d}$, where $\underline{d}$ and $\bar{d}$ are given bounds. Sample α must then be moved to resource $r(2)$. The travel times $w_{r(i),r(j)}$ $(0 \leqslant i, j \leqslant 2)$ between resources $r(i)$ and $r(j)$ are symmetrical. There are n identical samples and the objective is to minimize the time at which the last sample enters resource $r(2)$.

This problem can be described with six numbers: n, $\underline{d}$, $\bar{d}$, $w_{r(0),r(1)}$, $w_{r(1),r(2)}$ and $w_{r(0),r(2)}$. To our knowledge, the complexity of this problem is open. We even do not know whether there exists any algorithm, polynomial in n, for solving this problem. In order to show that this problem is less trivial than might be apparent, assume first (as in the classical literature) that the capacity of resource $r(1)$ is equal to 1. In that case, an optimal schedule would simply consist of completely treating the first sample, than the second, and so on. This best schedule S has the following makespan:

$$\text{makespan}(S) = n(w_{r(0),r(1)} + \underline{d} + w_{r(1),r(2)}) + (n-1)\,w_{r(0),r(2)}.$$

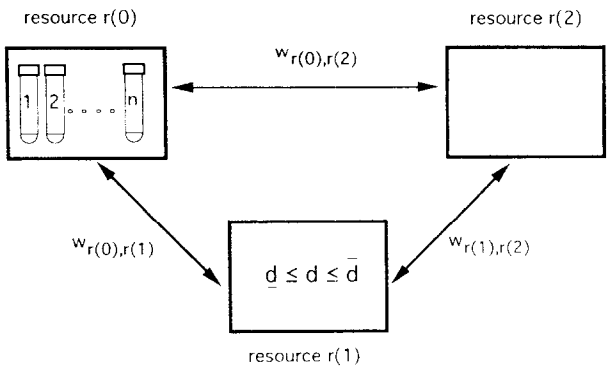

Fig. 1.

As an example, if $w_{r(0),r(1)} = 2$, $w_{r(1),r(2)} = 3$, $w_{r(0),r(2)} = 4$, $n = 4$ and $\underline{d} = 10$, we would get a schedule which makespan is equal to 72.

Assume now that the capacity of $r(1)$ is larger or equal to n and that $\underline{d} = \bar{d}$. Instead of waiting $\underline{d}$ time units at $r(1)$, the robot can move additional samples to $r(1)$. A first intuitive idea for constructing a schedule would consist of introducing the samples into $r(1)$ as early as possible.

Once a sample enters $r(1)$, the robot has to move to $r(0)$ and turn back to $r(1)$ for introducing an additional sample into $r(1)$. This takes $2w_{r(0),r(1)}$ time units. Moreover, once a sample leaves $r(1)$, the robot has to move to $r(2)$ and turn back to $r(1)$ for removing an additional sample from $r(1)$. Hence, since the duration of task 1 is fixed, at least $2w_{r(1),r(2)}$ time units should separate two placements of a sample into $r(1)$. In summary, the amount of time between two placements of a sample into $r(1)$ should at least be twice as large as the maximum value among $w_{r(0),r(1)}$ and $w_{r(1),r(2)}$.

By considering the same problem instance as above, a first sample is introduced into $r(1)$ at time $2(= w_{r(0),r(1)})$. The second sample enters $r(1)$ at time $8(= 2 + 2w_{r(1),r(2)})$. Since sample 1 has to leave $r(1)$ at time $12(= 2 + \underline{d})$, no additional sample can be introduced into $r(1)$ at this stage. The robot will therefore wait at $r(1)$ during 4 time units and move samples 1 and 2 to $r(2)$. The complete schedule for the $n = 4$ samples is represented in Fig. 2(a). Its makespan is equal to 46.

It can be worthwhile to wait a little bit before introducing a sample into $r(1)$. It is not difficult to prove that the best schedule for the problem instance studied above (where $c_{r(1)} \geqslant n$ and $\underline{d} = \bar{d} = 10$) consists of introducing sample 2 into $r(1)$ at time 11 instead of 8. While 3 time units are wasted at this stage, the makespan of the complete schedule is equal to 43, as can be observed in Fig. 2(b). Since at least 9 time units separate two placements of a sample into $r(1)$, the robot has enough time to introduce a new sample into $r(1)$ between two removals of a sample from $r(1)$.

Assume now that $\underline{d} < \bar{d}$ while the capacity of $r(1)$ is still larger or equal to n. In this case, the duration of task 1 is not fixed exactly. For the same problem instance as above, consider the values $\underline{d} = 10$ and $\bar{d} = 15$. The schedules represented in Figs. 2(a)

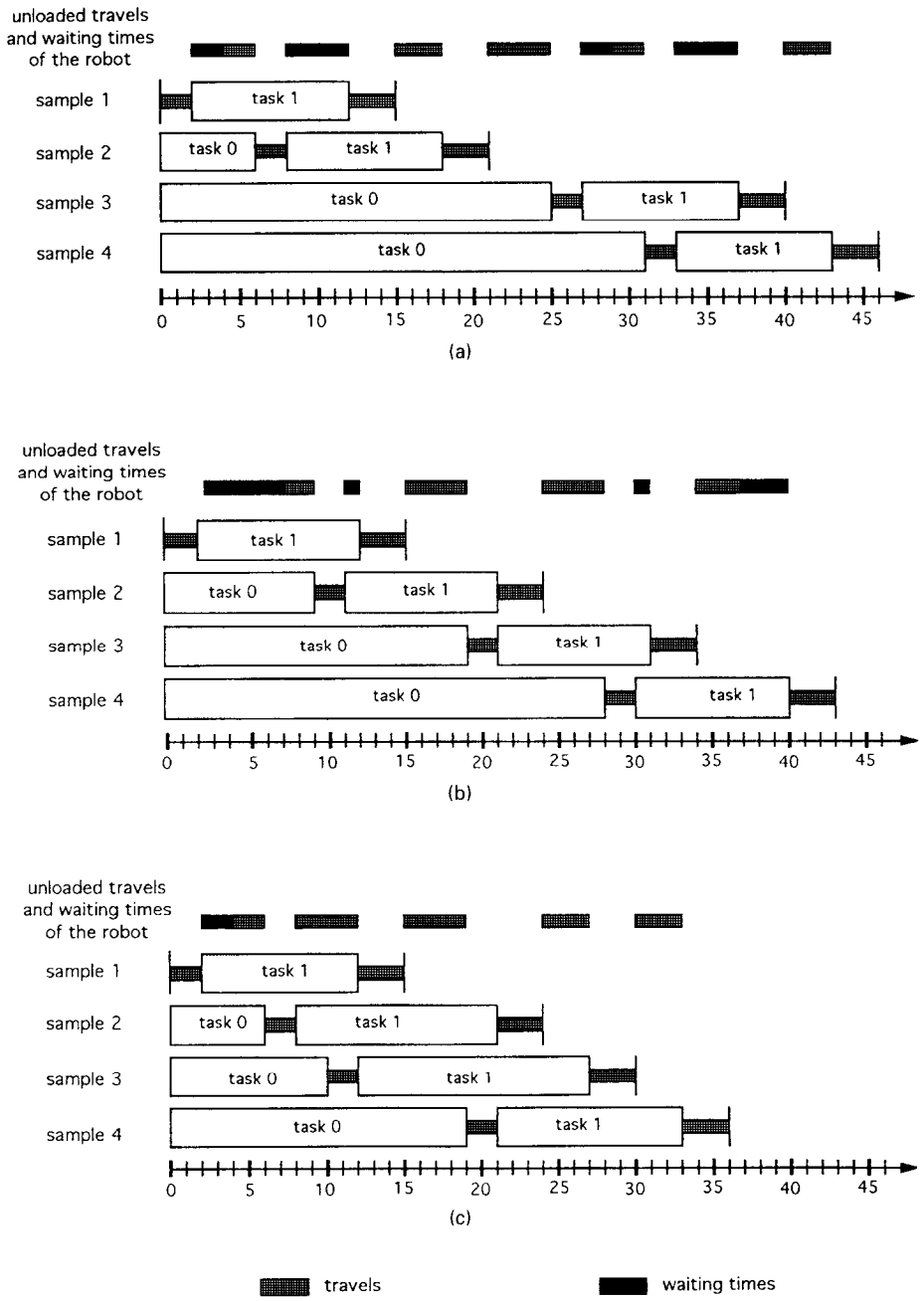

Fig. 2. (a) A schedule in 46 time units ($\underline{d} = \bar{d} = 10$). (b) A schedule in 43 time units ($\underline{d} = \bar{d} = 10$). (c) A schedule in 36 time units ($\underline{d} = 10$ and $\bar{d} = 15$).

and (b) are still feasible. However, the fact that task 1 may last more than $\underline{d}$ time units provides flexibility, and it turns out that a better schedule can be constructed in that case. Indeed, the schedule represented in Fig. 2(c) has a makespan equal to 36. It can be proved that it is optimal. As can be observed, the time spent by the samples 1–4 in $r(1)$ are, respectively, equal to 10, 13, 15 and 12. Hence, three samples among four have spent more than $\underline{d} = 10$ time units in $r(1)$.

Notice that the makespan of the schedule represented in Fig. 2(c) is twice as good as the schedule corresponding to the best solution of the classical problem studied in the literature.

At this point, we assume that the reader is convinced that the considered scheduling problem is less trivial than might be apparent. The schedule consisting of completely treating the first sample, then the second, and so on, is not necessarily feasible since the stay of the samples in $r(0)$ is eventually limited in time. We conjecture that finding a feasible schedule is an NP-hard problem. Therefore, we have developed a heuristic solution method which is described in the next section.

3. The heuristic scheduling algorithm

3.1. Basic approach and notations

The general step of an iterative procedure consists of constructing from a current solution s a new solution s' and in checking whether one should stop there or perform another step. Neighborhood search techniques (e.g. tabu search or simulated annealing) are iterative procedures where a neighborhood $N(s)$ is defined for each solution s and the next solution s' is selected from the solutions in $N(s)$. Such methods have successfully been applied to various scheduling problems (e.g. [2, 8, 16, 21, 28, 35]).

For adapting a neighborhood search technique to a specific combinatorial optimization problem, two basic ingredients must be defined: the set $\mathscr{S}$ of solutions which will be explored during the search, and the neighborhood $N(s)$ of each solution s in $\mathscr{S}$.

In our case, we cannot define $\mathscr{S}$ as the set of feasible schedules (i.e satisfying all the constraints) since finding a feasible schedule is possibly an NP-hard problem. Notice also that if $\mathscr{S}$ is the set of feasible schedules and s any solution in $\mathscr{S}$, it is not trivial to define the set $N(s)$ which should contain only feasible schedules.

The set $\mathscr{S}$ of solutions should therefore contain schedules which violate some constraints. For example, one could decide to relax the constraints on the duration of the active period of the tasks, each violation being penalized. Given an ordering of the tasks of the robot, it would then be difficult to decide how much time each task should last in order to minimize some measure of the violations.

It turns out, that neighborhood search techniques can hardly be applied to our scheduling problem. For this reason, we have developed a heuristic scheduling algorithm which is based on an iterative constructive method.

At the beginning of the algorithm, all samples are staying in the initial resource $r(0)$. At some intermediate step of the algorithm, a partial schedule, up to a time T, has been determined and all samples are at a certain stage of their chemical process. It is to be decided which task will be performed next. In other words, a partial schedule is characterized by the series of instructions that the robot has carried out thus far. The active periods of the already completed tasks are known. A partial schedule is augmented by an instruction to the robot to move a certain sample.

All partial schedules will take into account the capacity constraints and the fact that some resources are blocking. Hence, if the robot is staying in a blocking resource where a sample α has been introduced, then the robot has to wait, and its next task will necessarily consist of moving α to the next resource. Moreover, it will never be decided to move a sample to a resource $r(i)$ if the number of samples currently treated in $r(i)$ is equal to its capacity $c_{r(i)}$.

Let S be a partial schedule. In the following, we shall denote

- $r(i)$ the resource in which task i has to be performed,
- $R(S)$ the resource in which the robot is currently staying,
- $w_{r(i),r(j)}$ the time needed by the robot for moving from resource $r(i)$ to resource $r(j)$,
- $x_{\alpha,i}$ (resp $y_{\alpha,i}$) the time at which sample α enters (resp. leaves) resource $r(i)$,
- $b_{\alpha,i}$ (resp $e_{\alpha,i}$) the time at which the active period of task starts (resp. ends) on sample α,
- $\underline{d}_i$ (resp. $\bar{d}_i$) the minimal (resp. maximal) duration of the active period of task i (it is assumed that $\underline{d}_0 = 0$ and $\bar{d}_p = \infty$),
- $t_\alpha(S)$ the task currently performed on sample α,
- $s(S)$ the sample that the robot moved last,
- $T(S)$ the time at which the robot is ready to move for performing a new task (this time is equal to $x_{\alpha,t_\alpha(S)}$ where $\alpha = s(S)$),
- S_α the solution obtained from S by moving sample α to resource $r(t_\alpha(S) + 1)$.

3.2. The constraints

Let S be a partial schedule and let us assume that it has been decided to move a sample α from resource $r(t_\alpha(S))$ to resource $r(t_\alpha(S) + 1)$. In order to build the partial schedule of the next iteration, two moves of the robot have to be scheduled. We have to decide when the robot will leave $R(S)$ for reaching $r(t_\alpha(S))$, when it will remove α from $r(t_\alpha(S))$ and when it will introduce α into $r(t_\alpha(S) + 1)$.

Since the robot has to move from $R(S)$ to $r(t_\alpha(S))$ before removing α, the time $y_{\alpha,t_\alpha(S)}$ has to be larger or equal to $T(S) + w_{R(S),r(t_\alpha(S))}$. Denote $\gamma = s(S)$, $i = t_\gamma(S)$ and $j = t_\alpha(S)$. Hence, the last task performed by the robot was the placement of sample γ into $r(i)$. This was done at time $T(S) = x_{\gamma,i}$ and it has now been decided to remove sample α from $r(j)$. We have

$$y_{\alpha,j} \geqslant x_{\gamma,i} + w_{r(i),r(j)}.$$

The time between the removal of α from $r(j)$ and the placement of α into $r(j+1)$ must be larger or equal to the travel time from $r(j)$ to $r(j+1)$. Hence, we have

$$x_{\alpha,j+1} - y_{\alpha,j} \geqslant w_{r(j),r(j+1)}.$$

If the left part of the inequality is not equal to the right part, this means that the robot is used as a storage resource with α in its grip during $x_{\alpha,j+1} - y_{\alpha,j} - w_{r(j),r(j+1)}$ time units. This may happen if α should immediately be removed from $r(j)$ while its placement into $r(j+1)$ should not occur too early.

In summary, consider two consecutive tasks of the robot, the first one consisting of moving sample γ to resource $r(i)$ for performing task i, and the second one consisting of moving sample α from resource $r(j)$ to resource $r(j+1)$ for performing task $j+1$. We have the following constraint:

$$x_{\gamma,i} + w_{r(i),r(j)} \leqslant y_{\alpha,j} \leqslant x_{\alpha,j+1} - w_{r(j),r(j+1)}. \tag{1}$$

We have seen that the active period of a task t ($1 \leqslant t \leqslant p-1$) on a sample α is not necessarily the whole interval $[x_{\alpha,t}, y_{\alpha,t}]$. We have the following constraints:

$$x_{\alpha,t} \leqslant b_{\alpha,t}, \tag{2}$$

$$e_{\alpha,t} \leqslant y_{\alpha,t}, \tag{3}$$

$$b_{\alpha,t} + \underline{d}_t \leqslant e_{\alpha,t} \leqslant b_{\alpha,t} + \bar{d}_t. \tag{4}$$

In the case where $r(t)$ has an implicit activation, inequalities (2) and (3) are replaced by equalities.

Let $L_{i,j}$ denote the maximal amount of time separating the end of the active period of task i on a sample α and the beginning of the active period of task j on α. We have

$$b_{\alpha,j} \leqslant e_{\alpha,i} + L_{i,j}. \tag{5}$$

A partial schedule is called *feasible* if it satisfies constraints (1)–(5).

Notice that between the end of the active period of task i and the beginning of the active period of task j, sample α must be moved to resources $r(i+1), \ldots, r(j)$, while each task t ($i+1 \leqslant t < j$) lasts at least $\underline{d}_t$ time units. Hence, given any pair (i,j) of tasks such that $0 \leqslant i < j \leqslant p$, the following condition is trivially necessary for the whole problem to be feasible.

$$\sum_{t=i}^{j-1} w_{r(t),r(t+1)} + \sum_{t=i+1}^{j-1} \underline{d}_t \leqslant L_{i,j}.$$

A partial schedule S is completely characterized by the following information:

- the set $\{t_\alpha(S)\colon \alpha = 1, \ldots, n\}$ of tasks currently performed on the samples,
- the times $x_{\alpha,i}$, $b_{\alpha,i}$, $e_{\alpha,i}$ and $y_{\alpha,i}$ for all $\alpha = 1, \ldots, n$ and $i = 1, \ldots, t_\alpha(S) - 1$,
- the times $y_{\alpha,0}$ for all $\alpha = 1, \ldots, n$ such that $t_\alpha(S) > 0$,
- the times $x_{\alpha,t_\alpha(S)}$ for all $\alpha = 1, \ldots, n$.

For the initial solution S, we have $t_\alpha(S) = 0$ and $x_{\alpha,0} = 0$ for all $\alpha = 1, \ldots, n$. Let S be a partial schedule at an intermediate step of the algorithm. Let α be any sample

which has not yet been introduced into the final resource $r(p)$ and denote $t = t_\alpha(S)$. We now compute the earliest and latest times at which α can leave $r(t)$. According to constraints (2)–(4), we have

$$y_{\alpha,t} \geqslant e_{\alpha,t} \geqslant b_{\alpha,t} + \underline{d}_t \geqslant x_{\alpha,t} + \underline{d}_t.$$

If the next instruction given to the robot is the removal of α from $r(t)$, the robot will first have to move to $r(t)$. It follows that $y_{\alpha,t} \geqslant T(S) + w_{R(S),r(t)}$. Therefore, the earliest time $\underline{h}_\alpha$ at which α can leave $r(t)$ is defined as follows:

$$\underline{h}_\alpha = \max\{x_{\alpha,t} + \underline{d}_t,\ T(S) + w_{R(S),r(t)}\}.$$

Let i be any task such that $i < t$. According to constraints (1),(2) and (5), we have the following inequalities:

$$e_{\alpha,i} + L_{i,t+1} \geqslant b_{\alpha,t+1} \geqslant x_{\alpha,t+1} \geqslant y_{\alpha,t} + w_{r(t),r(t+1)}.$$

Hence, we must have

$$y_{\alpha,t} \leqslant T_1 = \min_{i<t}(e_{\alpha,i} + L_{i,t+1}) - w_{r(t),r(t+1)}.$$

According to constraints (2)–(4), we know that if $r(t)$ has an implicit activation, then $y_{\alpha,t} \leqslant x_{\alpha,t} + \bar{d}_t$. If the activation of $r(t)$ is explicit, then the following inequalities follow from (4) and (5):

$$e_{\alpha,t} - \bar{d}_t \leqslant b_{\alpha,t} \leqslant e_{\alpha,i} + L_{i,t} \quad \text{for all } i < t,$$

$$b_{\alpha,t+1} \leqslant e_{\alpha,t} + L_{t,t+1}.$$

Hence, by (1) and (2), we must have (see Fig. 3)

$$\begin{aligned} y_{\alpha,t} &\leqslant x_{\alpha,t+1} - w_{r(t),r(t+1)} \\ &\leqslant b_{\alpha,t+1} - w_{r(t),r(t+1)} \\ &\leqslant \min_{i<t}(e_{\alpha,i} + L_{i,t}) + \bar{d}_t + L_{t,t+1} - w_{r(t),r(t+1)}. \end{aligned}$$

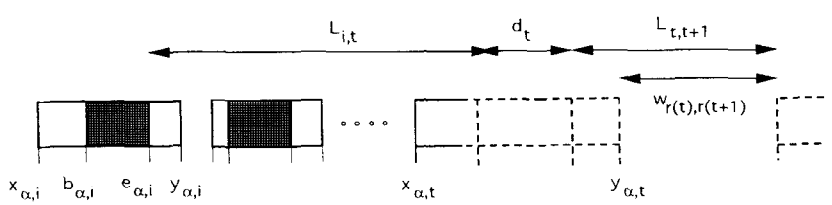

Fig. 3.

It follows that

$$y_{\alpha,t} \leqslant T_2 = \begin{cases} \min\limits_{i<t}(e_{\alpha,i} + L_{i,t}) + \bar{d}_t + L_{t,t+1} - w_{r(t),r(t+1)} \\ \quad \text{if } r(t) \text{ has an explicit activation,} \\ x_{\alpha,t} + \bar{d}_t \\ \quad \text{if } r(t) \text{ has an implicit activation.} \end{cases}$$

Therefore, the latest time $\bar{h}_\alpha$ at which sample α must leave $r(t)$ is defined as follows:

$$\bar{h}_\alpha = \min\{T_1, T_2\}.$$

In summary, we have the following inequalities:

$$\underline{h}_\alpha \leqslant y_{\alpha,t} \leqslant \bar{h}_\alpha. \tag{6}$$

Notice that if constraint (5) concerns pairs of consecutive tasks only, then $\bar{h}_\alpha$ can be computed in an easier way:

$$\bar{h}_\alpha = \begin{cases} e_{\alpha,t-1} + L_{t-1,t} + \bar{d}_t + L_{t,t+1} - w_{r(t),r(t+1)} \\ \quad \text{if } r(t) \text{ has an explicit activation,} \\ x_{\alpha,t} + \bar{d}_t \\ \quad \text{if } r(t) \text{ has an implicit activation.} \end{cases}$$

At each step of the algorithm we shall only consider feasible partial schedules which satisfy the following constraint:

$$\underline{h}_\alpha \leqslant \bar{h}_\alpha \quad \text{for all } \alpha = 1, \ldots, n. \tag{7}$$

Let S be a feasible partial schedule which satisfies constraint (7). For moving from S to S_α, we shall execute the following procedure, where t denotes task $t_\alpha(S)$:

Procedure MOVE (α)

$y_{\alpha,t} \leftarrow \underline{h}_\alpha$

$e_{\alpha,t} \leftarrow \min\{h_\alpha, \min\limits_{i<t}(e_{\alpha,i} + L_{i,t}) + \bar{d}_t\}$

$b_{\alpha,t} \leftarrow \min\{e_{\alpha,t} - \underline{d}_t, \min\limits_{i<t}(e_{\alpha,i} + L_{i,t})\}$

$x_{\alpha,t+1} \leftarrow y_{\alpha,t} + w_{r(t),r(t+1)}$

End of MOVE

In words:

(i) sample α leaves resource $r(t)$ as soon as possible (i.e. at time $\underline{h}_\alpha$);

(ii) in order to keep flexibility, the end $e_{\alpha,t}$ of the active period of t is set as late as possible (given (i)). It will then be easier to extend the partial schedule while satisfying constraint (5);

(iii) the duration $(e_{\alpha,t} - b_{\alpha,t})$ of the active period of t is chosen as short as possible (given (ii));
(iv) sample α is introduced into resource $r(t+1)$ as soon as possible.

The departure of the robot from $R(S)$ is scheduled at time $y_{\alpha,t} - w_{R(S),r(t)}$. This means that the robot waits at resource $R(S)$ between times $T(S)$ and $y_{\alpha,t} - w_{R(S),r(t)}$.

It may happen at some step of the algorithm that no extended solution S_α from S satisfies constraint (7). However, this does not mean that the ordering of the tasks of the robot in the feasible partial schedule S must be changed. Bad choices have perhaps been taken when giving values to the variables $x_{\beta,i}$, $b_{\beta,i}$, $e_{\beta,i}$ and $y_{\beta,i}$ ($\beta = 1,\ldots,n$; $i = 0,\ldots,t_\beta(S)$).

Consider for example the following instance of the problem represented in Fig. 1: $n = 2$, $\underline{d}_1 = 5$, $\bar{d}_1 = 6$, $w_{r(0),r(1)} = 2$, $w_{r(1),r(2)} = 3$ and $w_{r(0),r(2)} = 5$. There are no constraints of type (5) and it is assumed that $\bar{d}_0 = \infty$.

For the initial solution S, we have $t_1(S) = t_2(S) = 0$, $x_{1,0} = x_{2,0} = 0$, $\underline{h}_1 = \underline{h}_2 = 0$ and $\bar{h}_1 = \bar{h}_2 = \infty$. Without loss of generality, we may assume that sample 1 is chosen at the first iteration. By executing MOVE(1) we get a new partial schedule with $y_{1,0} = 0$ and $x_{1,1} = 2$. Moreover $\underline{h}_1 = 7$, $\bar{h}_1 = 8$, $\underline{h}_2 = 4$ and $\bar{h}_2 = \infty$.

At iteration 2, we may decide to move sample 1 to resource $r(2)$. In such a case, we call MOVE(1) and we will then have to execute MOVE(2) twice. A complete schedule of value 25 is obtained at iteration 4. This schedule is represented in Fig. 4(a).

By choosing sample 2 at the second iteration and executing MOVE(2), we get a partial schedule with $y_{2,0} = 4$, $x_{2,1} = 6$, $\underline{h}_1 = 7$, $\bar{h}_1 = 8$, $\underline{h}_2 = 11$ and $\bar{h}_2 = 12$. Now, we must choose sample 1 since $\bar{h}_1 = 8 < 11 = \underline{h}_2$. By calling MOVE(1), we get a partial schedule with $y_{1,1} = 7$ and $x_{1,2} = 10$. This solution does not satisfy constraint (7) since $\bar{h}_2 = 12 < \underline{h}_2 = 13$ (see Fig. 4(b)).

This does not mean however that the robot should not first move both samples 1 and 2 from $r(0)$ to $r(1)$. Indeed, such a decision leads to an optimal solution S^* with $T(S^*) = 16$. This optimal schedule is represented in Fig. 4(c).

By comparing the optimal solution S^* with the partial schedule of Fig. 4(b), we notice that we should have given value 7 instead of 6 to the variable $x_{2,1}$. It is shown in the next section how such a decision could have been made in polynomial time.

3.3. A longest path problem

All constraints of type (1)–(5) request that the difference value between two variables has to be larger or equal to a given value. Such constraints are called *difference constraints* [7]. A *constraint graph* $G = (V, A)$ can be associated with any system $Ax \geqslant b$ of difference constraints. The vertex set consists of a vertex v_i for each unknown variables x_i, plus an additional vertex v_0 which guarantees that every other vertex is reachable from it. The arc set A contains

- an arc (v_0, v_i) of length 0 for each unknown x_i of the system;
- an arc (v_i, v_j) of length b_k for each difference constraint $x_j - x_i \geqslant b_k$.

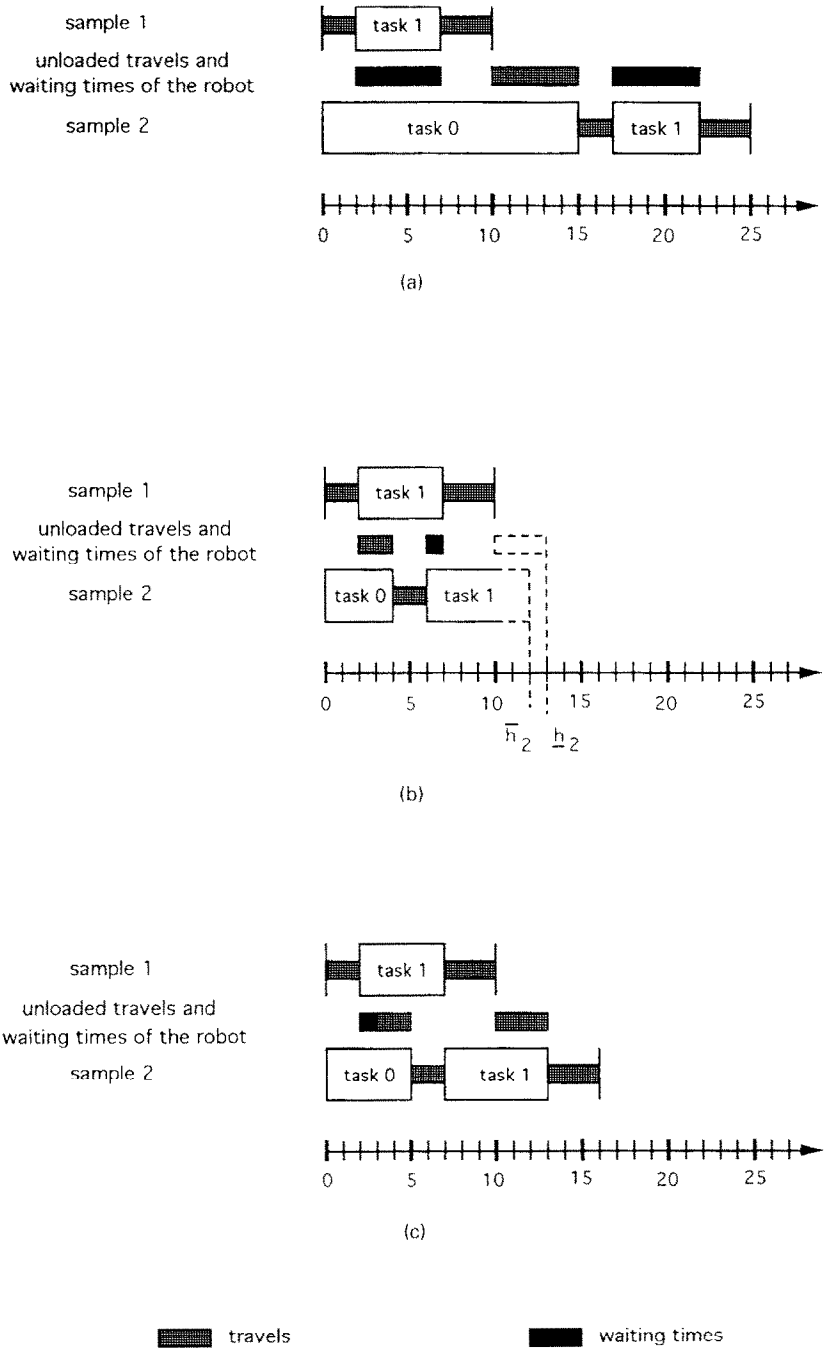

Fig. 4. (a) A schedule in 25 time units. (b) A partial solution that does not satisfy constraint (7). (c) A schedule in 16 time units.

It is well known (see [18] or [7, Ch. 25] for more details) that a system $Ax \geqslant b$ of difference constraints and its associated constraint graph G satisfy the following properties:

If G contains a circuit of positive length, then the system has no feasible solution.

If G contains no circuit of positive length, then a solution of the system can be obtained by giving to each unknown x_i the value of the longest path from v_0 to v_i in G.

Given an ordering of the tasks of the robot up to some iteration, deciding whether these tasks can be scheduled while satisfying constraints (1)–(5) and (7) is equivalent to testing whether a system of difference constraints has a feasible solution. The associated constraint graph $G = (V, A)$ is described below.

Let X, B, E and Y denote the following sets:

$$X = \{x_{\alpha,i}\colon \alpha = 1,\dots,n \text{ and } i = 1,\dots,\min\{p, t_\alpha(S)+1\}\},$$

$$B = \{b_{\alpha,i}\colon \alpha = 1,\dots,n \text{ and } i = 1,\dots,\min\{p-1, t_\alpha(S)\}\},$$

$$E = \{e_{\alpha,i}\colon \alpha = 1,\dots,n \text{ and } i = 1,\dots,\min\{p-1, t_\alpha(S)\}\},$$

$$Y = \{y_{\alpha,i}\colon \alpha = 1,\dots,n \text{ and } i = 0,\dots,\min\{p-1, t_\alpha(S)\}\}.$$

The vertex set V of the graph G consists of the union of X, B, E and Y and an additional vertex $\mathfrak{a}$.

There are four different types of arcs:

Type 1: For each sample α, the following arcs represent the stay of α in the initial resource:

$\text{arc}(\mathfrak{a}, y_{\alpha,0})$ of length 0,

$\text{arc}(y_{\alpha,0}, \mathfrak{a})$ of length $-\bar{d}_0$.

Type 2: For each sample α and each task i $(1 \leqslant i \leqslant t_\alpha(S))$ we have the following arcs which represent constraints (2)–(4):

$\text{arc}(x_{\alpha,i}, b_{\alpha,i})$ of length 0,

$\text{arc}(b_{\alpha,i}, e_{\alpha,i})$ of length $\underline{d}_i$,

$\text{arc}(e_{\alpha,i}, b_{\alpha,i})$ of length $-\bar{d}_i$,

$\text{arc}(e_{\alpha,i}, y_{\alpha,i})$ of length 0.

Moreover, if $r(i)$ has an implicit activation, the following arcs impose that the active period of task i on sample α is the whole interval $[x_{\alpha,t}, y_{\alpha,t}]$:

$\text{arc}(b_{\alpha,i}, x_{\alpha,i})$ of length 0,

$\text{arc}(y_{\alpha,i}, e_{\alpha,i})$ of length 0.

Type 3: For each sample α and for each pair (i,j) $(0 \leqslant i < j \leqslant \min\{p, t_\alpha(S)+1\})$ of tasks such that $L_{i,j} < \infty$, we have one of the following arcs representing constraint (5):

$\text{arc}(b_{\alpha,j}, e_{\alpha,i})$ of length $-L_{i,j}$ if $i > 0$ and $j < \min\{p, t_\alpha(S)+1\}$,

$\text{arc}(x_{\alpha,j}, e_{\alpha,i})$ of length $-L_{i,j}$ if $i > 0$ and $j = \min\{p, t_\alpha(S)+1\}$,

$\text{arc}(b_{\alpha,j}, y_{\alpha,i})$ of length $-L_{i,j}$ if $i = 0$ and $j < \min\{p, t_\alpha(S) + 1\}$,

$\text{arc}(x_{\alpha,j}, y_{\alpha,i})$ of length $-L_{i,j}$ if $i = 0$ and $j = \min\{p, t_\alpha(S) + 1\}$.

Type 4: The arcs of this type represent the moves of the robot.

- The transport of a sample α between two resources $r(i)$ and $r(i+1)$ $(0 \leqslant i < t_\alpha(S))$ is represented by the following arc:

$\text{arc}(y_{\alpha,i}, x_{\alpha,i+1})$ of length $w_{r(i),r(i+1)}$.

- Consider two consecutive tasks of the robot, the first one consisting of moving sample γ to resource $r(i)$ for performing task i $(1 \leqslant i \leqslant t_\gamma(S))$ and the second one consisting of moving sample α between resources $r(j)$ and $r(j+1)$ for performing task j $(1 \leqslant j \leqslant t_\alpha(S))$. The following arc represents the unloaded travel from resource $r(i)$ to resource $r(j)$:

$\text{arc}(x_{\gamma,i}, y_{\alpha,j})$ of length $w_{r(i),r(j)}$.

- Let α be any sample such that $t_\alpha(S) < p$. Denote $\gamma = s(S)$ and $t = t_\alpha(S)$. The following arcs correspond to the potential next two moves of the robot:

$\text{arc}(x_{\gamma,t(S)}, y_{\alpha,t})$ of length $w_{R(S),r(t)}$,

$\text{arc}(y_{\alpha,t}, x_{\alpha,t+1})$ of length $w_{r(t),r(t+1)}$.

This completely defines the constraint graph $G = (V, A)$. As mentioned above, the graph G contains a circuit of positive length if and only if the tasks of the robot can be scheduled in the given order while satisfying constraints (1)–(5) and (7) (i.e. the set of difference constraints).

If G has no circuit of positive length, then a feasible partial schedule denoted S_G can be obtained by giving to each variable $x_{\alpha,i}$, $b_{\alpha,i}$, $e_{\alpha,i}$, and $y_{\alpha,i}$ the value of the longest path from α to the corresponding vertex in G. This setting corresponds to the earliest possible schedule.

Notice that the use of the constraint graph is twofold. It is used not only for the detection of partial schedules which cannot be extended to a feasible complete schedule, but also for minimizing the makespan for a given sequence of robot instructions.

Finding a longest path between two vertices in a directed graph $G = (V, A)$ is a problem which is solvable in polynomial time. We use Bellman's $O(|A||V|)$ algorithm [3] which either computes the longest path from vertex α to all other vertices in the graph, or else finds a circuit of positive length.

As an example, consider the ordered set of tasks of the partial schedule represented in Fig. 4(b). The associated constraint graph is given in Fig. 5. The longest path from α to $x_{2,1}$ is represented in bold lines. Its length is equal to 7, which means that sample 2 should not be introduced into $r(1)$ before time 7 (as noticed in Section 3.2). Since the graph does not contain any circuit of positive length, a feasible partial schedule can be obtained by giving to each variable $y_{1,0}$, $x_{1,1}$, $b_{1,1}$, $e_{1,1}$, $y_{1,1}$, $x_{1,2}$, $y_{2,0}$ and $x_{2,1}$ the

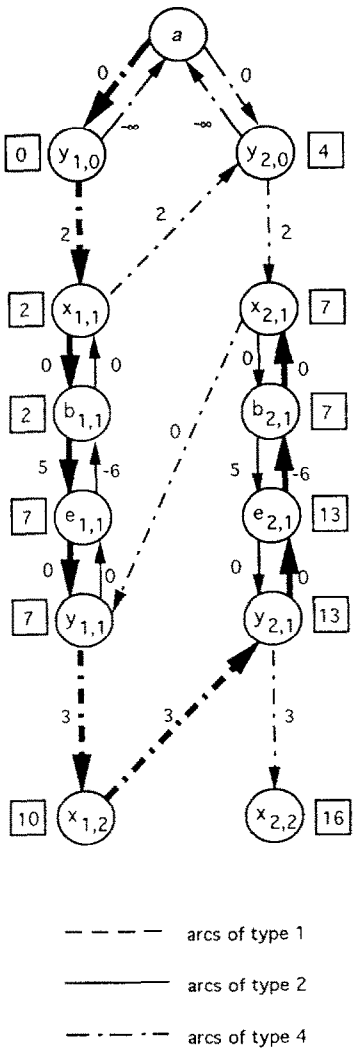

Fig. 5.

value of the longest path from a to the corresponding vertex. The numbers in the boxes represent these values.

3.4. *A kind of backtracking*

We now have to describe what happens when a circuit of positive length is detected. In such a case, the partial ordering of the tasks of the robot must be changed. We have decided to remove all robot instructions involving a sample α (chosen with some rule).

In other words, all tasks performed on sample α are cancelled and the remainder of the ordering is not changed. The graph corresponding to this new ordering is denoted $G - \{\alpha\}$ and is obtained from G by performing the following changes.

Construction of the reduced graph $G - \{\alpha\}$

Update of the moves of the robot

(a) Consider any path P of the following form ($x \to y$ denotes an arc (x, y)):

$$x_{\beta,i} \to y_{\alpha,j} \to x_{\alpha,j+1} \to y_{\alpha,j+1} \to x_{\alpha,j+2} \to \cdots \to x_{\alpha,j+h} \to y_{\delta,k}$$

where $\beta \neq \alpha$ and $\delta \neq \alpha$.

The path P is replaced by an arc $x_{\beta,i} \to y_{\delta,k}$ of length $w_{r(i),r(k)}$

(b) remove all arcs of type 4 involving sample α

(c) Let β be the last sample not equal to α which was moved in S. In the partial schedule corresponding to the new ordering, the robot is staying in resource $r(t_\beta(S))$. The next instruction which will be given to the robot may be the move of sample α between resources $r(0)$ and $r(1)$. Hence, the following arcs must be added:

- arc $(x_{\beta,t_\beta(S)}, y_{\alpha,0})$ of length $w_{r(t_\beta(S)),r(0)}$
- arc $(y_{\alpha,0}, x_{\alpha,1})$ of length $w_{r(0),r(1)}$

Removal of the remaining arcs and vertices concerning sample α

(d) remove all arcs of type 2 and 3 involving sample α

(e) remove vertices $b_{\alpha,i}$, $e_{\alpha,i}$, $y_{\alpha,i}$ $(1 \leqslant i \leqslant t_\alpha(S))$ and $x_{\alpha,i}$ $(2 \leqslant i \leqslant \min\{p, t_\alpha(S) + 1\})$

Consider the same example as in Fig. 4, with the only difference that $w_{r(1),r(2)} = 4$ (instead of 3). Assume that the first three tasks of the robot are the move of samples 1 and 2 from resource $r(0)$ to resource $r(1)$ and then the placement of sample 1 into the final resource $r(2)$. The graph G corresponding to the partial ordering made of these three tasks is represented in Fig. 6(a). It contains a circuit of length 2 represented in bold lines.

By removing sample 2 from the partial ordering, we get the graph $G - \{2\}$ of Fig. 6(b) which does not contain any circuit of positive length. The partial schedule $S_{G-\{2\}}$ can now be completed by calling MOVE(2) twice.

3.5. *Description of the algorithm*

Assume that the graph G corresponding to a partial ordering of the tasks of the robot contains a circuit of positive length. By removing all tasks concerning a sample α, it may happen that the reduced graph $G - \{\alpha\}$ still contains a circuit of positive length. In such a case, an additional sample is removed, and this process is repeated until the reduced graph does not contain any circuit of positive length. The rule for choosing the sample to be removed will be defined later. We can now describe the scheduling algorithm.

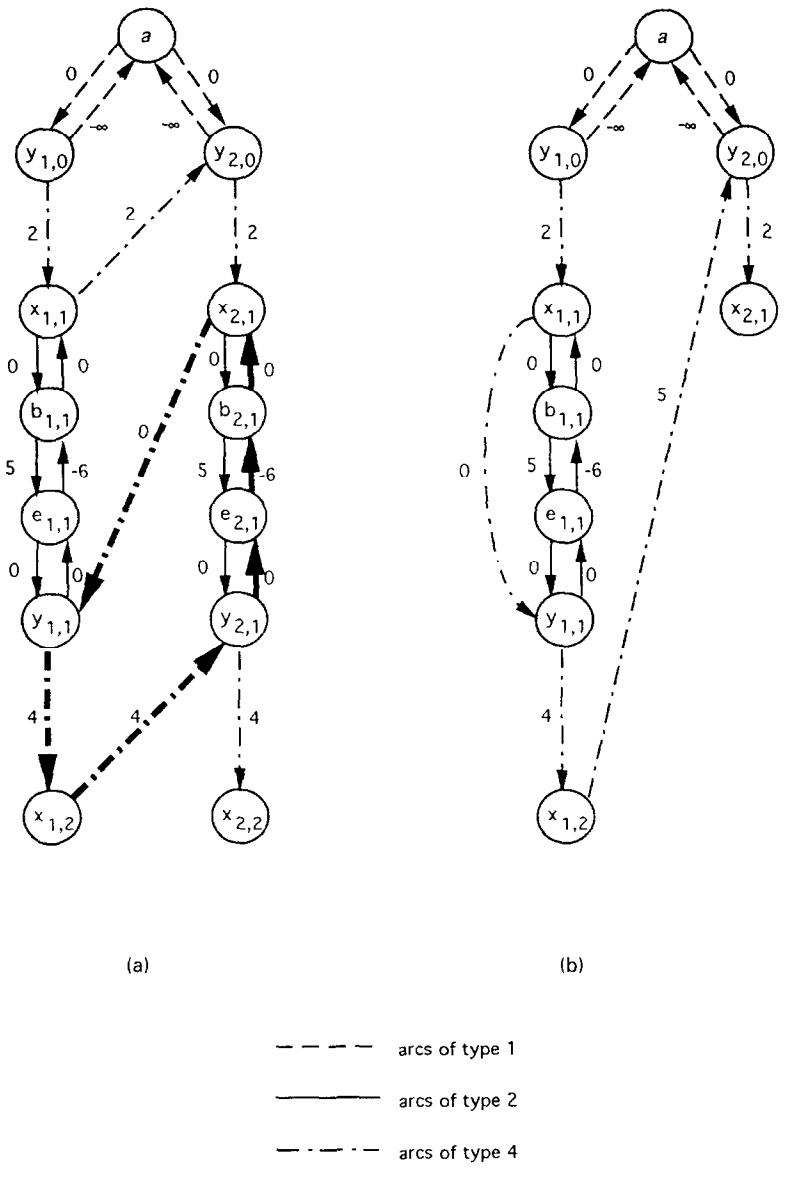

Fig. 6. (a) The graph G contains a circuit of positive length. (b) The reduced graph $G - \{2\}$.

The algorithm

Step 0: Initialize the partial schedule S by setting

$x_{\alpha,0} \leftarrow 0$; $t_\alpha(S) \leftarrow 0$; $\underline{h}_\alpha \leftarrow 0$; $\overline{h}_\alpha \leftarrow \overline{d}_0$; for $\alpha = 1, \ldots, n$

$c \leftarrow 0$; (* iteration counter *)

Step 1: Based on some rule $\mathfrak{R}_1$, choose a sample α with $t_\alpha(S) < p$

Step 2: set $c \leftarrow c + 1$;
call MOVE(α) ; set $S \leftarrow S_\alpha$ and update $\underline{h}_\beta$ and $\bar{h}_\beta$ for $\beta = 1, \ldots, n$;
if S is a feasible partial schedule satisfying constraint (7) then go to step 7
Step 3: Let O be the partial ordering induced by S.
Construct the constraint graph G associated with the ordering O.
Step 4: Use Bellman's algorithm for either computing the longest path from α to all other vertices in G, or else for finding a circuit of positive length in G; If G contains no circuit of positive length then set $S \leftarrow S_G$ and go to step 7;
Step 5: Based on some rule $\mathfrak{R}_2$, choose a sample β with $t_\beta(S) > 0$;
Step 6: Set $c \leftarrow c - t_\beta(S)$ and $G \leftarrow G\text{-}\{\beta\}$;
go to step 4
Step 7: If $c = n \cdot p$ then STOP else go to step 1

In order to complete the description of the algorithm, we now define rules $\mathfrak{R}_1$ and $\mathfrak{R}_2$. Consider a partial schedule S. We first define a set A of samples (candidates to be moved at the next iteration) by taking into account the limited capacity of the resources and the fact that some resources are blocking. A resource is *free* if the number of samples currently in it is strictly smaller than its capacity.

If the resource $R(S)$ in which the robot is currently staying is a blocking resource, then define $A = \{s(S)\}$ (because rule $\mathfrak{R}_1$ has to choose sample $s(S)$).

Otherwise, all blocking resources are free. Let q_α ($t_\alpha(S) < q_\alpha < p$) denote the task such that every $r(t)$ ($t_\alpha(S) < t < q_\alpha$) is a blocking resource and $r(q_\alpha)$ is not blocking. In other words, if the robot removes sample α from resource $r(t_\alpha(S))$, it will then have to move α to resources $r(t_\alpha(S) + 1), \ldots, r(q_\alpha)$ before moving any other sample. We define A as the set of samples α such that resource $r(q_\alpha)$ is free.

If resource $r(1)$ is free, then only one sample α with $t_\alpha(S) = 0$ (if any) is put into A (since all samples are equivalent).

We now describe two additional ingredients. The first one provides, at each iteration, an upper bound on the number of samples which are candidates to be moved. First notice that the size of the set A increases with the number of samples which are not in the initial or the final resource. When A is of large size, it becomes difficult to guess which move should be performed next in order to obtain a partial schedule which can be extended to a complete feasible solution. We have observed that steps 5 and 6 of the algorithm (which can be viewed as a kind of backtracking) are frequently performed when too many samples are out of $r(0)$ and $r(p)$ at the same time. For this reason, an upper bound N is defined on the total number of samples which are not in the initial or the final resource. We only consider feasible partial schedules S which satisfy the following constraint:

$$|\{\alpha: 0 < t_\alpha(S) < p\}| \leqslant N. \tag{8}$$

Hence, rule $\mathfrak{R}_1$ has to choose a sample in a set of candidates of size at most N. In the case where $|\{\alpha: 0 < t_\alpha(S) < p\}| = N$ and A contains a sample α with $t_\alpha(S) = 0$, then α is

removed from A since performing task 1 on α would provide a partial schedule which violates constraint (8).

The second ingredient prevents cycling to some extent. We define a matrix M of size $n \times p$ in which each element $m_{\alpha,i}$ ($1 \leqslant \alpha \leqslant n$; $0 \leqslant i < p$) stores the last value assigned by the algorithm to $y_{\alpha,i}$; initially $m_{\alpha,i}$ is set equal to a negative value.

Let α be any sample of A. If there exists a sample β ($1 \leqslant \beta \leqslant n$) such that $m_{\beta,t_\alpha(S)} = \underline{h}_\alpha$ then α is removed from A. The reason is that $y_{\alpha,t_\alpha(S)}$ would be set equal to $\underline{h}_\alpha$ in S_α, and the end of task $t_\alpha(S)$ has already been scheduled at that time (possibly for a sample $\beta \neq \alpha$) at an earlier stage of the algorithm.

Now, if A is empty, then we stop the algorithm. Otherwise, let α be any sample of A. As already noticed, S_α does not necessarily satisfy constraint (7). Let V_α denote the set of samples for which constraint (7) is violated in S_α. We select a subset B of A according to one of the following definitions:

(a) $B = \{\alpha \in A: |V_\alpha| = \min_{\beta \in A} |V_\beta|\}$,

(b) $B = \{\alpha \in A: \sum_{\delta \in V_\alpha} t_\delta(S) = \min_{\beta \in A} \sum_{\delta \in V_\beta} t_\delta(S)\}$,

(c) $B = \{\alpha \in A: \sum_{\delta \in V_\alpha} (\underline{h}_\delta - \bar{h}_\delta) = \min_{\beta \in A} \sum_{\delta \in V_\beta} (\underline{h}_\delta - \bar{h}_\delta)\}$,

(d) $B = \{\alpha \in A: \sum_{\delta \in V_\alpha} (\underline{h}_\delta - \bar{h}_\delta)/|V_\alpha| = \min_{\beta \in A} \sum_{\delta \in V_\beta} (\underline{h}_\delta - \bar{h}_\delta)/|V_\beta|\}$

(e) $B = \{$one sample of A chosen at random$\}$.

In words, let α be any sample for B:
- with (a), the number of samples violating constraint (7) in S_α is minimized;
- the idea behind definition (b) is that among all samples of V_α, we prefer to remove the one that is the less advanced in its chemical process;
- the difference $(\underline{h}_\delta - \bar{h}_\delta)$ for a sample δ in V_α is a measure of the violation of constraint (7); this means that by using definition (c) (resp. (d)), we minimize the total (resp. average) amount of violation in S_α.

Now, if B contains more than one sample, we then select a subset C of B according to one of the following definitions:

(f) $C = \{\alpha \in B: T(S_\alpha) = \min_{\beta \in B} T(S_\beta)\}$,

(g) $C = \{\alpha \in B: (y_{\alpha,t_\alpha(S)} - w_{R(S),R(S_\alpha)} - T(S)) = \min_{\beta \in B} (y_{\beta,t_\beta(S)} - w_{R(S),R(S_\beta)} - T(S))\}$,

(h) $C = \{$one sample of B chosen at random$\}$.

With (f), we minimize the time at which the robot is ready again for performing the next task. The waiting time of the robot is minimized by using definition (g).

Finally, if C contains more than one sample, then one of them is chosen at random. The variants for rule $\mathfrak{R}_1$ which have been tested are definition (e) and all combinations of one definition in $\{(a),(b),(c),(d)\}$ with one definition in $\{(f),(g),(h)\}$. According to the

results obtained on a real problem described in Section 4, it turns out that the best variants are the combinations of (a) or (b) with (g).

For a solution S_α, let D be the set of samples β with $0 < t_\beta(S_\alpha) < p$. We have tested the three following variants for rule $\mathfrak{R}_2$:

(i) choose the sample β in D which maximizes $y_{\beta,0}$ (the time at which β left $r(0)$),
(ii) choose a sample β in D at random,
(iii) choose a sample β in V_α at random.

Rule (iii) has given the best results for the problem test which is described in the next section.

4. Computational results

4.1. A real life problem

The proposed heuristic scheduling algorithm has been tested on a real life problem: the robotized sample preparation of membrane fatty acid esters for the identification of bacteria.

There are 12 resources in the laboratory with characteristics summarized in Table 1. The robotized analytical system is represented in Fig. 7. The ordered set of tasks which must be performed on each sample is described in Fig. 8. The minimal and maximal durations of the tasks as well as the constraints of type (5) are given in Table 2. The later constraints are for consecutive tasks only.

The travel times $w_{r(i),r(j)}$ are given in Table 3. It can be observed that the terms in the diagonal are not necessarily equal to zero. The reason is the following. When the robot is moved from a resource $r(i)$ to a resource $r(j)$, it is either for the removal of a sample from $r(j)$ or for the placement of a sample into $r(j)$. Let $\mathcal{T}_{\text{in}}(j)$ (resp. $\mathcal{T}_{\text{out}}(j)$) denote the time needed by the robot for introducing a sample into resource $r(j)$ (resp. removing a sample from $r(j)$). In case of a removal (resp. placement) of a sample, the

Table 1
Description of the resources

Resource	Type of activation	Capacity	Blocking
Refrigerator I	Implicit	∞	No
Dispenser I	Implicit	1	Yes
Dispenser II	Implicit	1	Yes
Dispenser III	Implicit	1	Yes
Dispenser IV	Implicit	1	Yes
Dispenser V	Implicit	1	Yes
Bath 100	Implicit	15	No
Bath 80	Implicit	15	No
Bath 20	Implicit	15	No
Rotator	Implicit	20	No
Vortex	Explicit	1	No
Refrigerator II	Implicit	∞	No

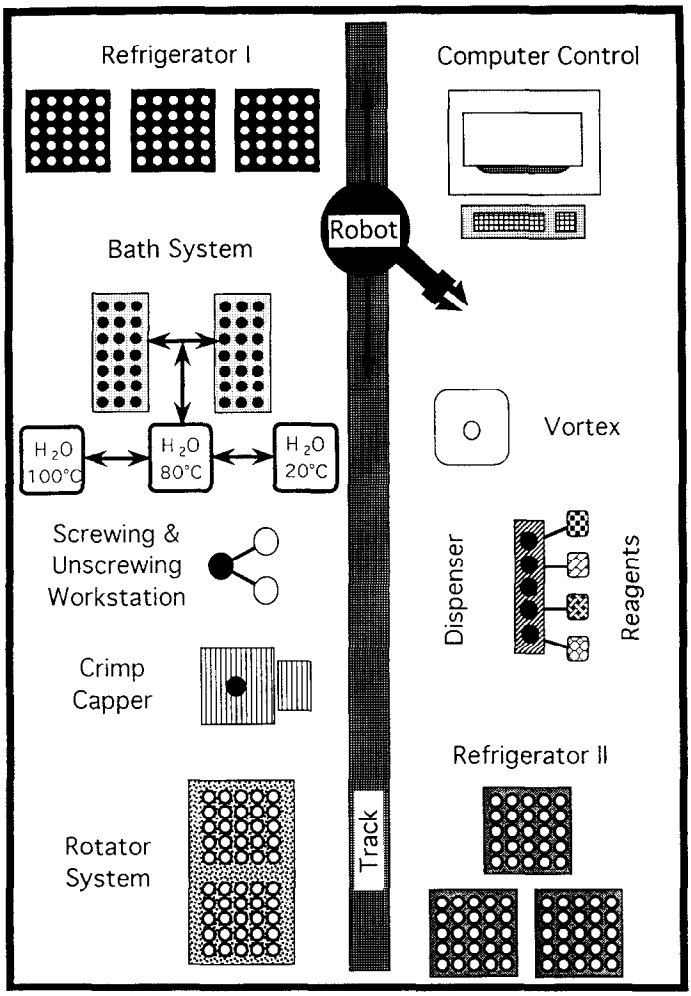

Fig. 7.

travel time $w_{r(i),r(j)}$ is equal to the effective time needed by the robot for moving from $r(i)$ to $r(j)$, plus $\mathcal{T}_{\text{in}}(i)$ (resp. $\mathcal{T}_{\text{out}}(i)$) time units. Now, let S be a partial schedule and assume that it has been decided to move a sample from $r(j)$ to $r(j+1)$. If the robot is already staying in resource $r(j)$ (that is, $R(S) = r(j)$), then the travel time $w_{r(j),r(j)}$ from $R(S) = r(j)$ to $r(j)$ may be larger than zero since it is equal to $\mathcal{T}_{\text{out}}(j)$.

We have tested 13 variants for rule $\mathfrak{R}_1$: (a)–(f), (a)–(g), (a)–(h), (b)–(f), (b)–(g), (b)–(h), (c)–(f), (c)–(g), (c)–(h), (d)–(f), (d)–(g), (d)–(h) and (e).

These variants have been combined with the three variants of rule $\mathfrak{R}_2$ (hence this gives a total amount of 39 variants).

Table 2
Ordered set of tasks which must be performed on each sample (all times are in seconds)

Task number i	Resource $r(i)$	Minimal duration $\underline{d}_i$	Maximal duration $\overline{d}$	$L_{i,i+1}$
0	Refrigerator I	0	∞	∞
1	Dispenser I	90	90	∞
2	Vortex	5	10	∞
3	Bath 100°	285	315	120
4	Vortex	5	10	∞
5	Bath 100°	1425	1575	120
6	Bath 20°	120	600	∞
7	Dispenser II	90	90	∞
8	Vortex	5	10	∞
9	Bath 80°	600	660	120
10	Bath 20°	120	600	∞
11	Dispenser III	90	90	∞
12	Rotator	600	900	∞
13	Dispenser IV	135	135	∞
14	Rotator	300	600	∞
15	Dispenser V	165	165	∞
16	Refrigerator II	0	∞	∞

Table 3
Travel times of the robot (in seconds)

$r(i)$	$r(j)$											
	1	2	3	4	5	6	7	8	9	10	11	12
1	3	3	3	3	3	3	9	10	10	10	8	7
2	0	0	0	0	0	0	5	5	5	6	5	3
3	0	0	0	0	0	0	5	5	5	6	5	3
4	0	0	0	0	0	0	5	5	5	6	5	3
5	0	0	0	0	0	0	5	5	5	6	5	3
6	0	0	0	0	0	0	5	5	5	6	5	3
7	6	5	5	5	5	5	6	8	8	9	8	6
8	7	5	5	5	5	5	8	6	8	9	8	6
9	7	5	5	5	5	5	8	8	6	8	9	5
10	8	7	7	7	7	7	10	10	9	7	10	6
11	7	7	7	7	7	7	10	10	11	11	8	5
12	4	3	3	3	3	3	6	6	5	5	3	0

Let S be a complete feasible schedule of the treatment of n samples. In order to perform the same treatment on n additional samples, the robot first has to move from $r(p)$ to $r(0)$. The combination of S and the move of the robot from $r(p)$ to $r(0)$ defines a cycle $\mathscr{C}$ which can be repeated indefinitely. Let $m(S)$ denote the makespan of S (i.e. the time at which the last sample enters resource $r(p)$). The solution value $f(S)$ of S is defined as follows:

$$f(S) = \frac{m(S) + w_{r(p),r(0)}}{n}.$$

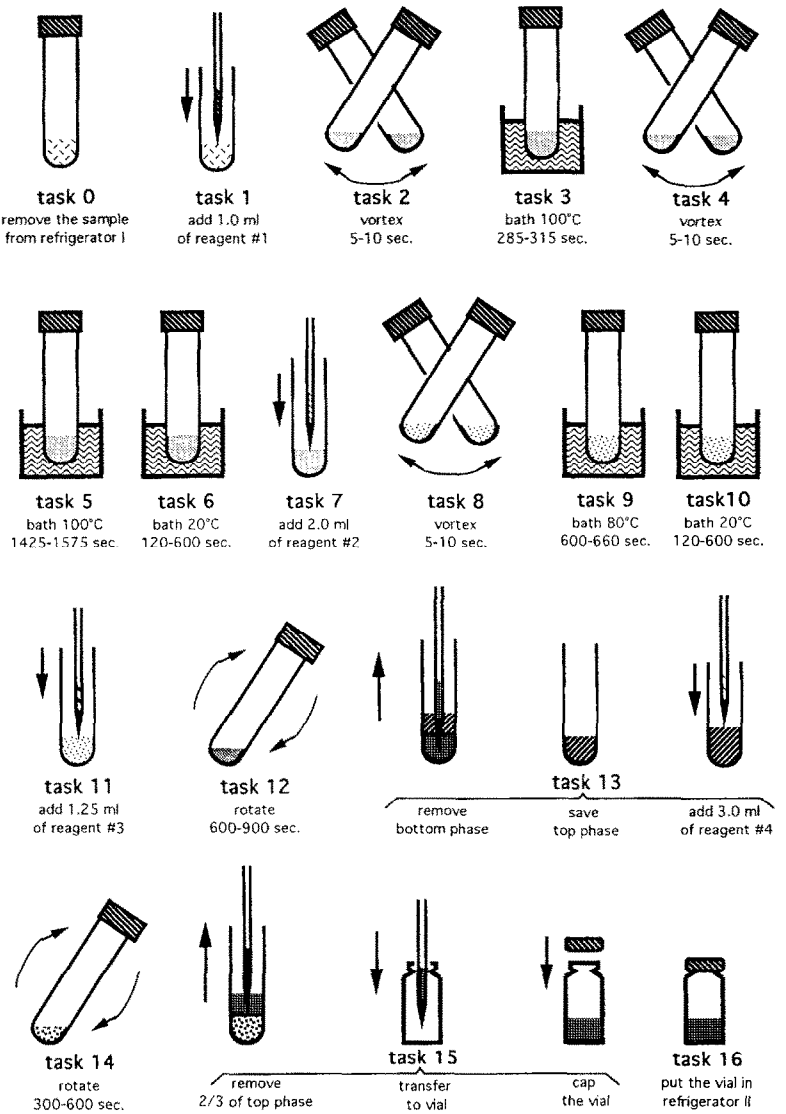

Fig. 8.

Hence, $f(S)$ is equal to the average time it takes to treat one sample in a repeating sequence with cycle $\mathscr{C}$. Minimizing this value is equivalent to maximizing the average throughout (number of samples moving to the final resource per time unit).

The sequence of tasks performed by the robot should not differ each time a new set of samples enters the laboratory, while the number of samples which have to be treated may vary from day to day and is usually not known in advance. For these

reasons, the number of samples treated in one cycle $\mathscr{C}$ should not be too large. In order to determine a cycle of appropriate size, we propose the following procedure.

Let $\bar{n}$ be an upper bound on the number of samples which can be treated in one cycle $\mathscr{C}$. Let $S^*(\bar{n}, N)$ be the schedule obtained by using the proposed algorithm on $\bar{n}$ samples with a given upper bound N on the number of samples which can be simultaneously out of $r(0)$ and $r(p)$. Let $O^*(\bar{n}, N)$ denote the ordering of the tasks of the robot induced by the schedule $S^*(\bar{n}, N)$. Given any number $n \leqslant \bar{n}$ of samples, a complete schedule $S(n, N)$ for n samples is computed as follows. We first consider the ordering $O(n, N)$ obtained from $O^*(\bar{n}, N)$ by removing all robot instructions involving samples $n+1, n+2, \ldots, \bar{n}$. We then construct the graph $G(n, N)$ associated with the ordering $O(n, N)$ of the tasks of the robot (for the treatment of n samples). Since $S^*(\bar{n}, N)$ is feasible, it follows that $G(n, N)$ does not contain any circuit of positive length. The schedule $S(n, N)$ is obtained by giving to each variable $x_{\alpha,i}$, $b_{\alpha,i}$, $e_{\alpha,i}$, and $y_{\alpha,i}$ the value of the longest path from a to the corresponding vertex in $G(n, N)$ (hence, $S(n, N) = S_{G(n,N)}$). By construction, the makespan of $S(n, N)$ is smaller or equal to the time needed for treating the first n samples according to the schedule $S^*(\bar{n}, N)$.

For our real life problem, the number of samples which are treated each day may vary from 100 to 1000. It has been decided to set $\bar{n} = 50$. In Fig. 9, the value of $S(n, N)$ $(n = 1, \ldots, 50)$ is compared with the value of the schedule obtained from $S^*(50, N)$ by removing all robot instructions concerning samples $n+1, \ldots, \bar{n} = 50$. All these values have been obtained by setting $N = 9$ and by using combination (a)–(g)–(iii). A best schedule of value 948 is reached for $n = 31$ while $f(S^*(50, 9)) = f(S(50, 9)) = 1015$.

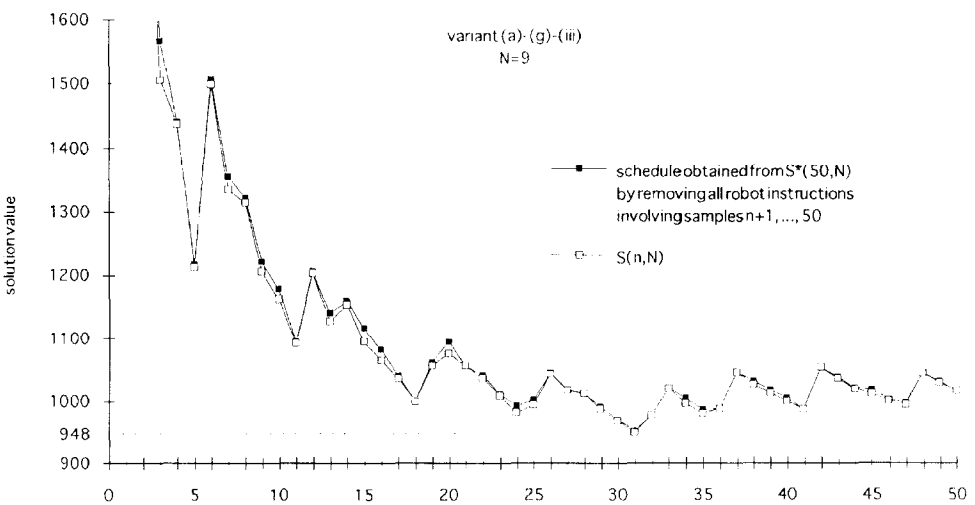

Fig. 9.

Let $F(\bar{n}, N)$ be the value of the best schedule among $S(1, N), \ldots, S(\bar{n}, N)$. Hence, we have

$$F(\bar{n}, N) = \min_{n=1,\ldots,\bar{n}} f(S(n, N)).$$

In our example, we have observed that $F(50, 9) = f(S(31, 9))$. In order to compare the 39 variants of the proposed algorithm we have computed the values $F(\bar{n}, N)$ with $N = 1, \ldots, \bar{n}$. It appeared clearly that definition (iii) should be used for rule $\mathfrak{R}_2$. As an example, we give in Fig. 10 the results obtained by combining the variant (a)–(g) of rule $\mathfrak{R}_1$ with the three possible variants of rule $\mathfrak{R}_2$. These three curves are representative of all other possible choices for rule $\mathfrak{R}_1$. Furthermore, we have observed that definition (g) for rule $\mathfrak{R}_1$ clearly dominates definitions (f) and (h). For example, the results obtained with combinations (d)–(f)–(iii), (d)–(g)–(iii) and (d)–(h)–(iii) are represented in Fig. 11. We got similar results by choosing (a), (b) or (c) instead of (d).

We have then combined (g)–(iii) with definitions (a)–(d). The results are reported in Fig. 12. A best schedule of value 942 could be determined by setting $N = 7$ and by using combination (a)–(g)–(iii) or (b)–(g)–(iii).

Finally, we have observed that definition (e) should not be used since the value of the best schedule that we could get with this variant of rule $\mathfrak{R}_1$ was equal to 3688.

Denote S_{best} the best schedule of value 942 which was found by the proposed algorithm. For this best schedule, we have represented in Fig. 13 the evolution of the

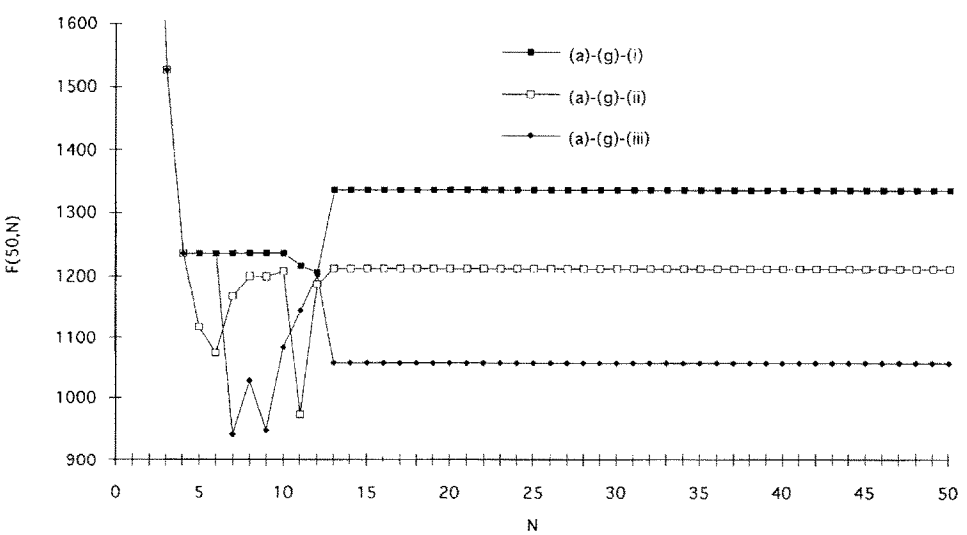

Fig. 10.

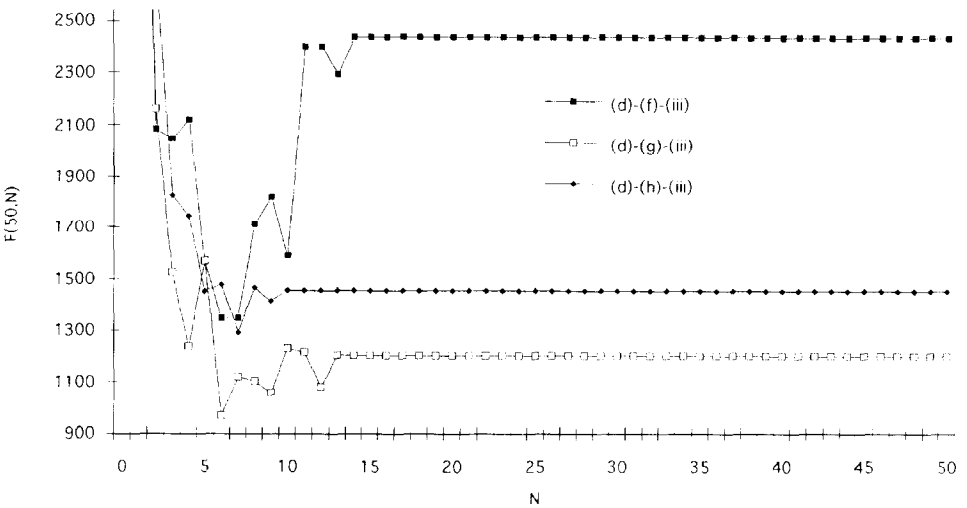

Fig. 11.

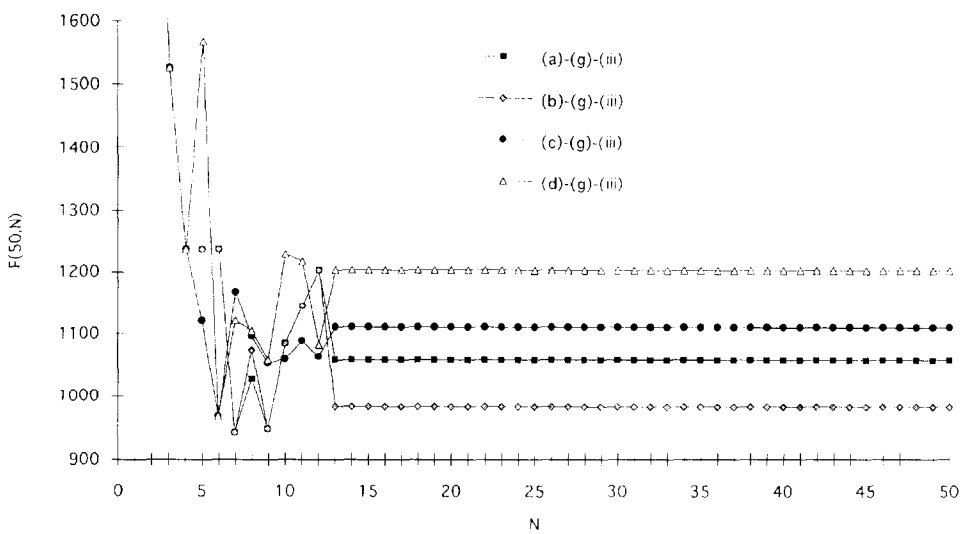

Fig. 12.

number of samples out of $r(0)$ and $r(p)$ as time increases. It can be observed that the part of the schedule between times 03:04:56 and 06:28:49 defines a cycle $\mathscr{C}$ during which 14 samples are removed from $r(0)$ while the same number of samples are introduced into $r(p)$. By running the proposed algorithm with $\bar{n}$ larger than 50, we would get a schedule where $\mathscr{C}$ is repeated as often as possible. Hence, if the number of samples which have to be treated is very large, we would get a schedule whose value

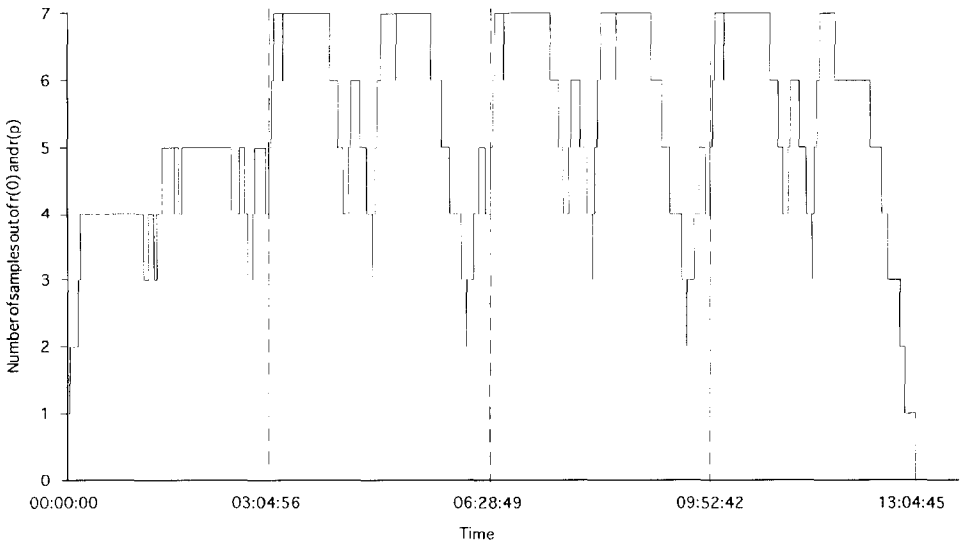

Fig. 13.

would tend to the average time needed to treat 14 samples according to $\mathscr{C}$, that is $(06:28:49-03:04:56)/14 = 874$ s.

A good lower bound on the optimal solution value of a schedule cannot easily be determined. One possible approach would be to ignore the robot (i.e. to set all travel times equal to zero). Because of the constraints of type (5) and the fact that the resources have a limited capacity which may be larger than one, this simplified problem does not seem to be much easier than the original one.

In our real life problem, the stay of the samples in the initial resource $r(0)$ is not limited in time (i.e. $\bar{d}_0 = \infty$). Hence, a feasible schedule of the treatment of n samples can easily be determined by computing $S^*(n, 1)$, that is, by completely scheduling the first sample, then the second one, and so on. The value of this schedule is $F(1, 1) = 4145$. Hence, the value of S_{best} is 4.4 times smaller than $F(1, 1)$. This is mainly due to the fact that, during the treatment of the samples according to S_{best}, there are in average 5.36 samples which are not in $r(0)$ or $r(p)$. In Fig. 14, we have represented the percentage of the time in S_{best} during which there is a given number of samples out of $r(0)$ and $r(p)$.

The quality of a schedule can also be measured by computing the part of the time during which the robot is waiting or performing unloaded moves. In S_{best}, unloaded travels occur in 4.92% of the schedule while the percentage of waiting time equals 19.26. For comparisons, the only unloaded travels in $S^*(n, 1)$ are performed from $r(p)$ to $r(0)$, each time the treatment of a sample has been completed. These represent 0.10% of the total time needed for treating n samples according to $S^*(n, 1)$. However, in this case, the total waiting time represents 81.88% of the schedule. This is due to the

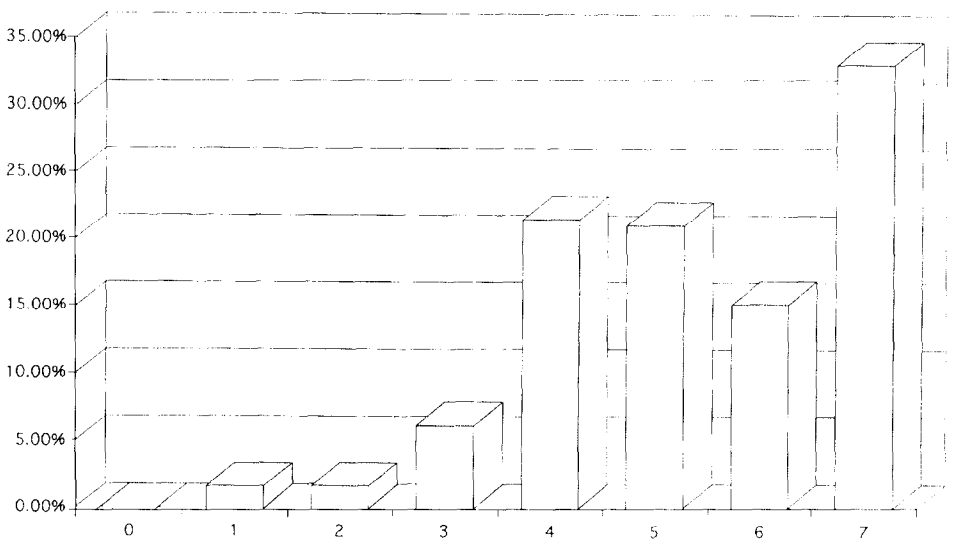

Fig. 14.

fact that each time a sample is introduced into a resource, the robot waits at this resource until the end of the task for moving the sample to the next resource.

4.2. Additional experiments

The most time consuming part of the proposed algorithm is step 4 where a longest path problem has to be solved. We have implemented a simplified version of the algorithm in which steps 3 and 4 are removed and steps 5 and 6 are replaced by the following ones:

Step 5: Remove all samples for which constraint (7) is violated (that is all β in V_γ),
Step 6: Set $c \leftarrow c - \sum_{\beta \in V_\gamma} t_\beta(S)$.

Call A_1 the original algorithm and A_2 its simplified version. The computational time is significantly reduced by using A_2 instead of A_1. But, of course, there is a loss in the quality of the solution. The solutions obtained with A_2 can easily be improved by solving a longest path problem at the end of the algorithm. The final solution of A_2 is thus compacted, all useless waiting times of the robot being removed. Therefore, an additional algorithm A_3 has been implemented which represents a kind of compromise between A_1 and A_2. For this third algorithm A_3, we have modified step 7 and added a step 8 as follows:

Step 7: If $c = n \cdot p$ then go to the step 8 else go to step 1,
Step 8: Let O be the partial ordering induced by S.
Construct the graph G induced by the ordering O.
Solve the longest path problem in G and set $S \leftarrow S_G$.
STOP

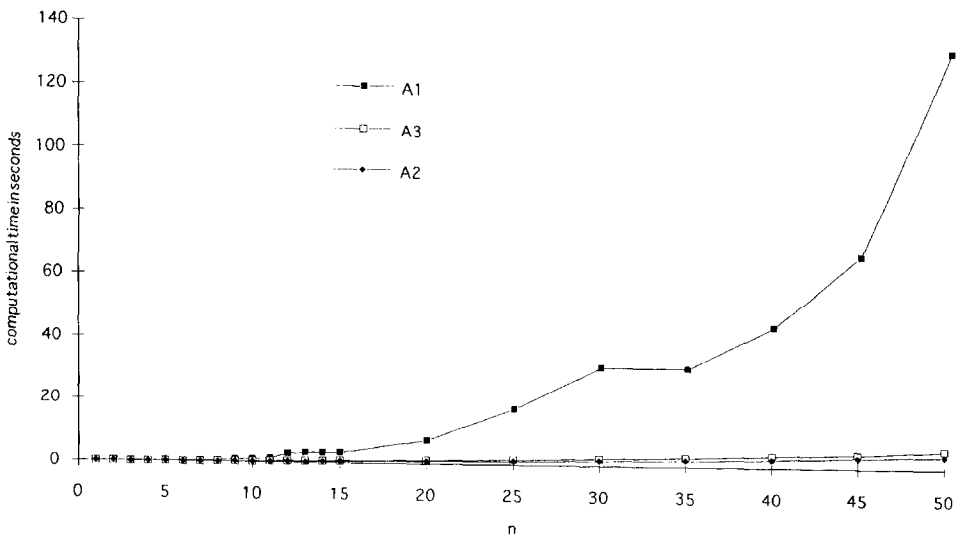

Fig. 15.

For solving longest path problems, we use Bellman's algorithm which runs in $O(|V||A|)$ where V is the vertex set and A the arc set of the graph. In our case, the number of vertices in the graph is proportional to the number v of samples which are not in $r(0)$, and the number of edges is proportional to v. Hence Bellman's algorithm runs in $O(v^2)$ with v increasing from 0 to n. This explains the observed complexity $O(n^3)$ for A_1. Performing steps 1–6 without solving a longest path problem requires a time in $O(n)$. Hence, A_2 and A_3 have both complexity $O(n^2)$, A_3 being slightly more time consuming than A_2. As an example, we have represented in Fig. 15 the computational times of the algorithms A_1, A_2 and A_3 obtained by setting $N = n$ with $n = 1, \ldots, 50$ and by using variant (a)–(g)–(iii). All the tests have been performed on a Silicon Graphics workstation.

The best schedule produced by A_2 is found by setting $n = 50$, by choosing N larger than 10 and by using combination (b)–(g)–(iii). This schedule of value 965 can then be improved by solving a longest path problem. A schedule of value 945 is obtained with this postoptimization. It corresponds to the best schedule which can be produced by using A_3.

These extensive computational experiments on a single problem instance may give hints for choosing good variants or rules $\mathfrak{R}_1$ and $\mathfrak{R}_2$. The scheduling problem on which the tests have been performed can also be considered as a benchmark problem for researchers interested in developing other heuristic (or exact) methods. In order to verify that the best variants mentioned above do not depend on our particular problem instance, we have performed additional tests on a random problem which is described in Tables 4–6. Once again, the best schedules are obtained by combining

Table 4
Description of the resources

Resource	Type of activation	Capacity	Blocking
1	Implicit	∞	No
2	Implicit	3	No
3	Explicit	10	No
4	Implicit	1	Yes
5	Implicit	6	No
6	Implicit	10	No
7	Implicit	1	Yes
8	Implicit	6	No
9	Implicit	6	No
10	Implicit	10	No
11	Explicit	3	No
12	Implicit	∞	No

Table 5
Ordered set of tasks which must be performed on each sample (all times are in seconds)

Task number i	Resources $r(i)$	Minimal duration $\underline{d}_i$	Maximal duration $\bar{d}$	$L_{i,i+1}$
0	1	0	∞	∞
1	2	148	291	∞
2	7	29	41	∞
3	5	118	295	17
4	11	80	156	∞
5	9	26	141	18
6	4	34	184	∞
7	5	165	580	30
8	9	137	790	22
9	8	112	264	∞
10	4	47	130	∞
11	7	35	177	∞
12	10	35	63	∞
13	2	142	190	26
14	3	102	179	∞
15	6	19	29	∞
16	12	0	∞	∞

(g)–(iii) with (a), (b), (c) or (d). The results that we got with these four combinations are represented in Fig. 16. A best schedule of value 285 is found by setting $N = 6$ and by using variant (d)–(g)–(iii). Moreover, the value of this best schedule is 4.5 times smaller than the value of $S^*(n, 1)$.

We could have generated many additional pseudo examples. These would certainly be far from real cases of robotized analytical systems and we would get tables of results which could not be compared with any other one.

Table 6
Travel times (in seconds) of the robot

$r(i)$	$r(j)$											
	1	2	3	4	5	6	7	8	9	10	11	12
1	0	2	5	3	2	3	3	2	5	4	5	5
2	2	0	2	4	3	1	3	1	5	2	1	1
3	5	2	0	3	3	3	3	5	2	3	4	4
4	3	4	3	0	1	4	1	4	5	2	5	1
5	2	3	3	1	0	3	5	3	3	4	1	1
6	3	1	3	4	3	0	2	3	5	3	3	5
7	3	3	3	1	5	2	0	1	5	1	5	3
8	2	1	5	4	3	3	1	0	1	2	2	4
9	5	5	2	5	3	5	5	1	0	4	2	3
10	4	2	3	2	4	3	1	2	4	0	5	3
11	5	1	4	5	1	3	5	2	2	5	0	5
12	5	1	4	1	1	5	3	4	3	3	5	0

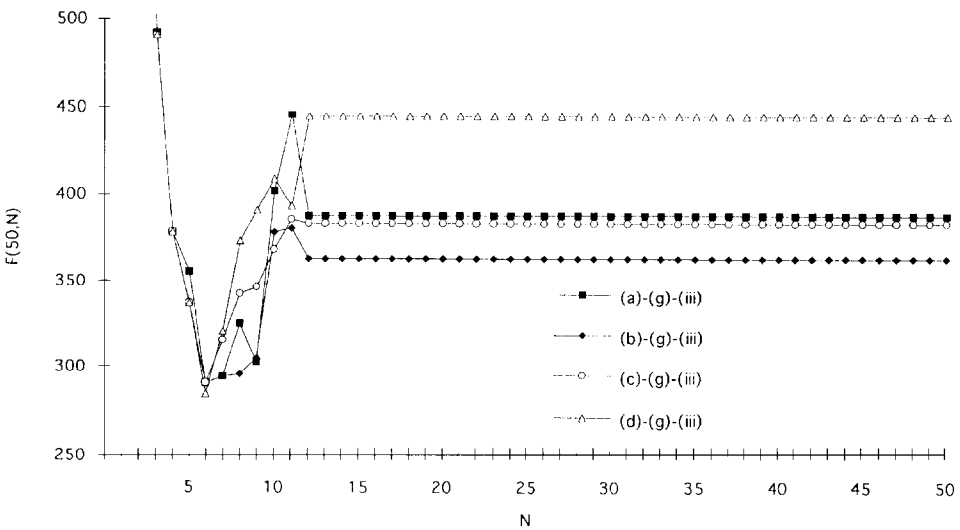

Fig. 16.

5. Extensions and final remarks

The model and algorithms presented in this paper are flexible in the sense that they can easily be extended for taking into account additional constraints.

It may happen that there are more than one resource in which a task can be performed. This occurs for example when there are identical resources. In such a case, let R_i be the set of resources in which task i may be performed. We would then define $r(\alpha, i)$ (instead of $r(i)$) as the resource of R_i in which task i is performed on sample α. Moreover, let S be a partial schedule, α a sample and let us denote $t = t_\alpha(S)$. We have

seen that $y_{\alpha,t}$ must be smaller or equal to the minimum of two latest times T_1 and T_2. When sample α is removed from $r(\alpha, t)$ we would then introduce it into the closest resource of R_{t-1} which is free. This means that, in the definitions of T_1 and T_2, the travel time $w_{r(t),r(t+1)}$ must be replaced by $\min_{r \in R_{t-1}} \{w_{r(\alpha,t),r}\}$.

All other changes are obvious. For example, if $R(S)$ is not a blocking resource, the set A of candidate samples to be moved at the next iteration is defined as the set of samples α such that at least one resource in R_{q_α} is free.

We have encountered problems in which places in resources may be reserved during some period for a specific sample. This may occur for example when the samples must visit a resource twice: part of the contents of the sample is extracted at the first visit, and restitution of the contents occurs at the second visit. It is important not to mix the contents of the samples. Hence, if sample α is put into place P of the resource for the extraction task, the restitution of the contents to α must occur at the same place P. This place P is reserved for α between these two tasks.

In such a case, we only have to change the definition of a free resource: we shall say that a resource r is free for a sample α if there exists a place in r which is not occupied and not reserved for any sample $\beta \neq \alpha$.

Future developments in this topic will consist of dealing with more complex systems containing storage resources, more than one robot, and in which different chemical treatments have to be performed simultaneously on several sets of samples.

References

[1] K.R. Baker, Introduction to Sequencing and Scheduling (Addison-Wesley, Reading, MA, 1974).

[2] J.W. Barnes and M. Laguna, A tabu search experience in production scheduling, Ann. Oper. Res. 41 (1993) 141–156.

[3] R.E. Bellman, On a routing problem, Quart. Appl. Math. 16 (1958) 87–90.

[4] J. Blazewicz, Selected topics in scheduling theory, Ann. Discrete Math. 31 (1987) 1–60.

[5] E.G. Coffman Jr, Computer & Job/Shop Scheduling Theory (Wiley, New York, 1976).

[6] R.W. Conway, W.L. Maxwell and L.W. Miller, Theory of Scheduling (Addison-Wesley, Reading, MA, 1967).

[7] T.H. Cormen, C.E. Leiserson and R.L. Rivest, Introduction to Algorithms (MIT Press, Cambridge, MA, 1990).

[8] M. Dell'Amico and M. Trubian, Applying tabu search to the job shop scheduling problem, Ann. Oper. Res. 41 (1993) 231–252.

[9] S. French, Sequencing and Scheduling (Wiley, New York, 1982).

[10] S. Goyal, Jobshop sequencing problem with no-wait in process, Internat. J. Prod. Res. 13 (1975) 197–206.

[11] S. Goyal and C. Sriskandarajah, No-wait shop scheduling: computational complexity and approximate algorithms. Opsearch 25 (1988) 220–244.

[12] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Ann. Discrete Math. 5 (1979) 287–326.

[13] N.G. Hall, H. Kamoun and C. Sriskandarajah, Scheduling in robotic cells: Two machine cells and identical parts, Working Paper 93-06, Department of Industrial Engineering, University of Toronto, Toronto, Ont. (1993).

[14] N.G. Hall, H. Kamoun and C. Sriskandarajah, Scheduling in robotic cells: Large cells, Working Paper 93-07, Department of Industrial Engineering, University of Toronto, Toronto, Ont. (1993).

[15] N.G. Hall and C. Sriskandarajah, Machine scheduling problems with blocking and no-wait in process, Working Paper 91–05, Department of Industrial Engineering, University of Toronto, Toronto, Ont. (1993).
[16] A. Hertz and M. Widmer, An improved tabu search approach for solving the job shop scheduling problem with tooling constraints, Discrete Appl. Math. 65 (1996) 319–345.
[17] I. Ioachim and F. Soumis, Schedule efficiency in a robotic production cell, Cahier du GERAD G-93-05, Ecole des HEC, Montréal, Que. (1993).
[18] J.E. Kelley Jr, Critical-path planning and scheduling: Mathematical basis, Oper. Res. 9 (1961) 296–320.
[19] H. Kise, T. Shioyama and T. Ibaraki, Automated two-machine flowshop scheduling: A solvable case, IIE Trans. 23 (1991) 10–16.
[20] W. Kubiak, A pseudo-polynomial algorithm for a two-machine no-wait jobshop scheduling problem, European J. Oper. Res. 43 (1989) 267–270.
[21] P.J.M. Laarhoven, E.H. Aarts and J.K. Lenstra, Job shop scheduling by simulated annealing, Oper. Res. 40 (1992) 113–125.
[22] E.L. Lawler, Recent results in the theory of machine scheduling, in: A. Bachem, M. Groetschel and B. Korte, eds., Mathematical Programming: The State of the Art (Springer, Berlin, 1982).
[23] E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Recent developments in deterministic sequencing and scheduling: A survey, in: M.A.H. Dempster et al., eds., Deterministic and Stochastic Scheduling (Reidel, Dordrecht, 1982).
[24] J.K. Lenstra and A.H.G. Rinnooy Kan, Scheduling theory since 1981: an annotated bibliography, in: M. O'Leigeartaigh, J.K. Lenstra and A.H.G. Rinnooy Kan, eds., Combinatorial Optimization: Annotated Bibliographies (Wiley, Chichester, 1985).
[25] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, Ann. Discrete Math. 1 (1977) 343–362.
[26] R. Logendran and C. Sriskandarajah, Sequencing of robot activities and parts in two machine robotic cells, Working Paper 91–06, Department of Industrial Engineering, University of Toronto, Toronto, Ont. (1992).
[27] P.L. Maggu and G. Dass, On $2 \times n$ sequencing problem with transportation time of jobs, Pure Appl. Math. Sci. 12 (1980) 1–6.
[28] E. Nowicki and C. Smutnicki, A fast taboo search algorithm for the job shop scheduling problem, Preprint 93/8, Instytut Cybernetyki Technicznej, Politechniki Wroclawskiej, Wroclaw (1993).
[29] S.S. Panwalkar and C.R. Woollam, Flowshop problems with no in-process waiting: a special case, J. Oper. Res. Soc. 30 (1979) 661–664.
[30] S.S. Panwalkar and C.R. Woollam, Ordered flowshop problems with no in-process waiting: further results, J. Oper. Res. Soc. 31 (1980) 1039–1043.
[31] S.S. Reddi and C.V. Ramamoorthy, On the flowshop sequencing problem with no-wait in process, Oper. Res. Quart. 23 (1972) 323–331.
[32] H. Rock, Some new results in no-wait flowshop scheduling, Z. Oper. Res. 28 (1984) 1–16.
[33] S. Sahni and Y. Cho, Complexity of scheduling shops with no-wait in process, Math. Oper. Res. 4 (1979) 448–457.
[34] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz and W. Kubiak, Sequencing of parts and robot moves in a robotic cell, Internat. J. Flexible Manufacturing Systems 4 (1992) 331–358.
[35] M. Widmer and A. Hertz, A new heuristic method for the flow shop sequencing problem, European J. Oper. Res. 41 (1989) 186–193.