Scheduling

The one-machine sequencing problem

Jacques CARLIER

Institut de programmation, Université Paris VI, 75230 Paris, France

Let M_1 and M_3 be non-bottleneck machines and M_2 a bottleneck machine processing only one job at a time. Suppose that n jobs have to be processed on M_1, M_2 and M_3 (in that order) and job i has to spend a time a_i on M_1, d_i on M_2 and q_i on M_3; we want to minimize the makespan.

This problem is important since its resolution provides a bound on the makespan of complicated systems such as job shops. It is NP-hard in the strong sense. However, efficient branch and bound methods exist and we describe one of them. Our bound for the tree-search is very close to the bound used by Florian et al., but the principle of branching is quite different. At every node, we construct by an $O(n \log n)$ algorithm a Schrage schedule; then we define a critical job c, a critical set J and consider two subsets of schedules: the schedules when c precedes every job of J and the schedules when c follows every job of J. We give the results of this method and prove that the difference between the optimum and the Schrage schedule is less than d_c.

1. Introduction

The pure sequencing problem is a specialized scheduling problem in which an ordering of the jobs completely determines a schedule. The simplest pure sequencing problem has a single resource or machine. As Baker [1] states, it is a building block in the development of a comprehensive understanding of complicated systems such as job shops.

We have to sequence n independent jobs on the machine; a job i is available for processing by the machine at the point in time a_i, has to spend an amount of time d_i on the machine, and an amount of time q_i in the system after its processing by the machine. Our objective is to minimize the makespan.

Numerous authors (Baker and Su [2], Bratley et al. [3], Florian et al. [5], Lageweg et al. [8]) studied this problem; Garey and Johnson [6] proved it to be NP-hard in the strong sense.

We study this problem and describe a branch and bound method to solve it. This method was coded and examples from 50 up to 10000 jobs were solved optimally.

2. General points

In this section we associate a conjunctive graph and a schedule with every sequence of jobs, then we state a theorem in order to provide lower bounds of the makespan.

Conjunctive graph. We associate with a sequence a conjunctive graph $G = (X, U)$. The set X of nodes is obtained by adding two fictitious nodes O and * to the set I of jobs: $X = I \cup \{O, *\}$; O is a job 'beginning', * a job 'end'. The set U of arcs includes three sets: $U = U_1 \cup U_2 \cup U_3$. Let $U_1 = \{(O, i) \mid i \in I\}$; arc$(O, i)$ is valued by a_i, so that job i cannot start before the point in time a_i. Let $U_2 = \{(i, *) \mid i \in I\}$; arc$(i, *)$ is valued by $q_i + d_i$, since job i has to spend an amount of time $q_i + d_i$ in the system after its beginning of processing by the machine. Let $U_3 = \{(i,j) \mid \text{job } i \text{ precedes job } j \text{ in the sequence}\}$; arc$(i,j)$ is valued by d_i; these arcs set the sequence.

Schedule. We associate with a sequence, the schedule $\mathfrak{I} = \{t_i \mid i \in X\}$ where the starting time t_i of job i is equal to the value of the maximal path from O to i in graph G. t_O is null and t_* is equal to the value of the critical path, in other words the makespan.

The aim is to find a sequence minimizing the value of the critical path in the associated conjunctive graph.

Proposition 1. *For all* $I_1 \subseteq I$,

$$h(I_1) = \underset{i \in I_1}{\text{Min}} \, a_i + \sum_{i \in I_1} d_i + \underset{i \in I_1}{\text{Min}} \, q_i$$

is a lower bound on the optimal makespan.

Proof. In the conjunctive graph associated with the optimal schedule, there is a path passing through O, then through every job of I_1, eventually through *; the value of this path is greater than $h(I_1)$ and lower than the value of the critical path that is equal to the optimal makespan. So $h(I_1)$ is lower than the optimal makespan.
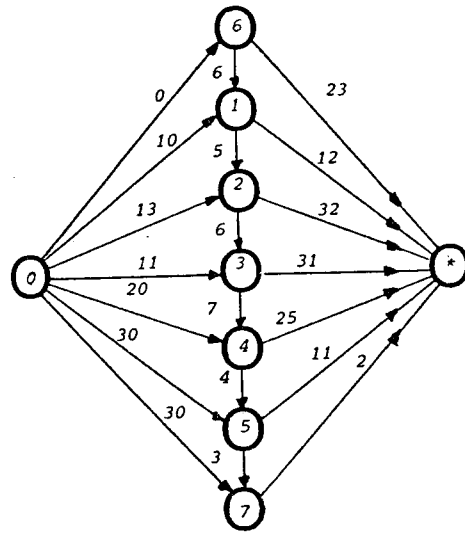
3. Study of Schrage algorithm

In the Schrage algorithm the job ready with the greatest q_i is scheduled. We present this algorithm and apply it to an example; then we prove that the distance to the optimum from the Schrage schedule is less than d_c where c is a critical job that we define; finally, we show how to code this algorithm with a complexity of $O(n \log n)$.

Schrage algorithm. In this algorithm, U is the set of jobs already scheduled and $\overline{U}$ is the set of all other jobs, t is the time.

  (i) Set $t = \underset{i \in I}{\text{Min}} \, a_i$; $U = \emptyset$.

  (ii) At time t, schedule amongst the 'ready' jobs i $(a_i \leqslant t)$ of $\overline{U}$, the job j with the greatest q_i (or any one in the case of ties).

  (iii) Set: $U = U \cup \{j\}$; $t_j = t$; $t = \text{Max}(t_j + d_j,$ $\underset{i \in \overline{U}}{\text{Min}} \, a_i)$; if U is equal to I, the algorithm is finished; otherwise, go to (ii).

The new value of t is the point in time when the machine becomes available for another job: job j is

Fig. 1. Associated conjunctive graph.

processed by the machine until $t_j + d_j$, and other jobs cannot be processed before $\underset{i \in \overline{U}}{\text{Min}} \, a_i$.

Example. We apply this algorithm to the example with seven jobs defined by Table 1.

The schedule built by the Schrage algorithm is: $t_1 = 10$, $t_2 = 15$, $t_3 = 21$, $t_4 = 28$, $t_5 = 32$, $t_0 = t_6 = 0$, $t_7 = 35$, $t_* = 53$; the associated conjunctive graph is drawn in Fig. 1.

The critical path is 0, 1, 2, 3, 4, *; its value is 53. So the makespan is equal to 53.

Theorem. *Let L be the makespan of the Schrage schedule.*

(a) *If this schedule is not optimal, there is a critical job c and a critical set J such that:*

$$h(J) = \underset{i \in J}{\text{Min}} \, a_i + \sum_{i \in J} d_i + \underset{i \in J}{\text{Min}} \, q_i > L - d_c.$$

Consequently, the distance to the optimum from the Schrage schedule is less than d_c; moreover, in an optimal schedule, either c will be processed before all

Table 1

Jobs i	1	2	3	4	5	6	7
Release dates a_i	10	13	11	20	30	0	30
Processing times d_i	5	6	7	4	3	6	2
Tails q_i	7	26	24	21	8	17	0

the jobs of J, or c will be processed after all the jobs of J.

(b) *If this schedule is optimal, there exists J such that* $h(J) = L$.

Proof. Let G be the conjunctive graph associated with the Schrage schedule; we consider a critical path μ of G passing through a maximal set of jobs. Generally, μ does not pass through every job of I.

We modify the numbering of jobs so that μ pass through $0, 1, \ldots, p, *$ in this order; hence the value of the critical path is $L = a_1 + \Sigma_{i=1,\ldots,p} d_i + q_p$.

First, we prove that in this schedule no job is processed between the times $t = a_1 - 1$ and $t = a_1$. If one job j was processed, it would be finished at time a_1, because the starting time of job 1 is $t_1 = a_1$; so $t_j + d_j = a_1$; then, $t_j + d_j + \Sigma_{i=1,\ldots,p} d_i + q_p = L$; there would be a critical path passing through $O, j, 1, \ldots, p, *$; this is in contradiction with the maximality of μ.

Second, we prove that $a_1 = \text{Min}_{i=1,\ldots,p} a_i$. We have just seen that the machine was idle between times $t = a_1 - 1$ and $t = a_1$; so, in the algorithm when job 1 is scheduled $a_1 = \text{Min}_{i \in \bar{U}} a_i$; and $a_1 = \text{Min}_{i=1,\ldots,p} a_i$, since $\{1, \ldots, p\} \subseteq \bar{U}$.

Third, *if* $q_p = \text{Min}_{i=1,\ldots,p} q_i$, the value of the critical path is:

$$L = a_1 + \sum_{i=1,\ldots,p} d_i + q_p$$

$$= \text{Min}_{i=1,\ldots,p} a_i + \sum_{i=1,\ldots,p} d_i + \text{Min}_{i=1,\ldots,p} q_i;$$

so $L = h(J)$ with $J = \{1, 2, \ldots, p\}$ and *the schedule is optimal.* This happens whenever $a_i < a_j$ implies $q_i \geq q_j$; for instance, it is the case when the a_i are all equal, or when the q_i are all equal (Jackson rule).

Otherwise, there exists $i < p$ such that $q_i < q_p$; let c be the greatest subscript such that $q_c < q_p$; we set $J = \{c+1, \ldots, p\}$; so $q_c < q_r$ for every $r \in J$. We verify that $r \in J$ implies $a_r > t_c$; if a_r was less than t_c, job r would have been ready and scheduled at time t_c since it would have had priority over job c ($q_c < q_r$).

Consequently, $r \in J$ implies $a_r > t_c = a_1 + d_1 + \cdots + d_{c-1}$; then $\text{Min}_{r \in J} a_r > a_1 + d_1 + \cdots + d_{c-1}$, and $q_p = \text{Min}_{r \in J} q_r$ implies $\text{Min}_{r \in J} a_r + d_{c+1} + \cdots + d_p + \text{Min}_{r \in J} q_r > a_1 + d_1 + \cdots + d_p + q_p - d_c$; the first term is the lower bound $h(J)$ on the makespan, the second term is
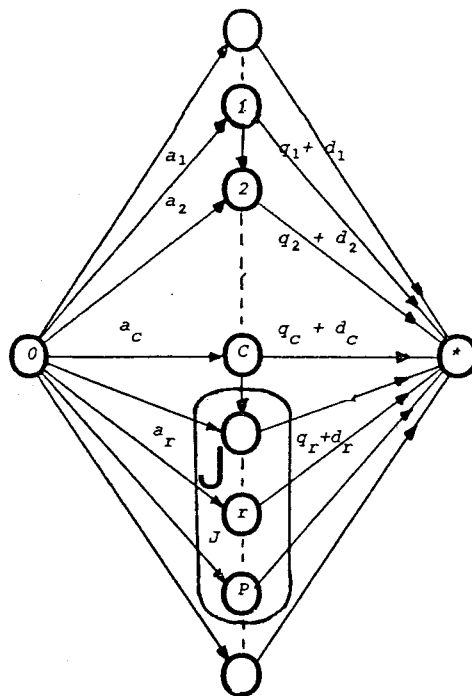

Fig. 2. Definitions of J and c.

$L - d_c$; so $h(J) > L - d_c$, and the distance to the optimum is less than d_c. Moreover, job c has to be processed either before all the jobs of J, or after all the jobs of J.
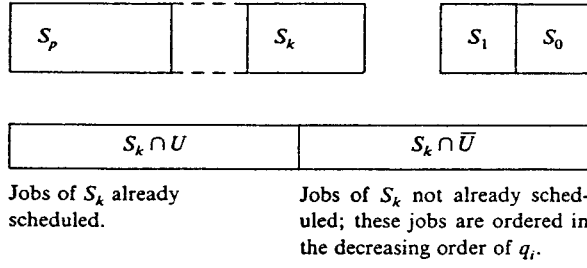
Remark. In the example (Fig. 1) $J = \{2, 3, 4\}$, $h(J) = 21 + 17 + 21 = 49$.

4. Programming of Schrage algorithm

We present an algorithm whose complexity is $O(n \log n)$. In this algorithm, S is the set of jobs such that $a_i \leq t$; m is the cardinal of S; $x_p \ldots x_1 x_0$ is the binary expression of m (p is chosen so that: $2^p \leq n = |I| < 2^{p+1}$). S will be partitionned into $p + 1$ sets, $S_p, \ldots, S_1, S_0$ the cardinals of which will be $2^p x_p, \ldots, 2^1 x_1, x_0$ (S_k is empty when $x_k = 0$).

Every non-empty set S_k will be ordered (see Fig. 3): first the jobs already scheduled, then the other jobs in the decreasing order of q_i. We define α_k by: α_k is null if S_k is empty or if all the jobs of S_k are scheduled, otherwise α_k is the subscript of the first job of S_k not already scheduled.

Every time a new job is introduced in S, we consider the empty set S_{k_0} with minimal subscript;

| S_p | | S_k | | S_1 | S_0 |

| $S_k \cap U$ | $S_k \cap \bar{U}$ |

Jobs of S_k already scheduled.

Jobs of S_k not already scheduled; these jobs are ordered in the decreasing order of q_i.

Fig. 3. Partition of S.

then, we set $S_{k_0} = \{m\} \cup S_0 \cup S_1 \cup \cdots \cup S_{k_0-1}$. In practice we use auxiliary sets M_k whose cardinals are 2^k and set: $M_0 = \{m\}$; $M_k = M_{k-1} \cup S_{k-1}$, for $k = 1$ to $k = k_0$; then: $S_{k_0} = M_{k_0}$.

Modified algorithm.

(i) *Initialisations.* We number the jobs in the increasing order of a_i, and set $t = \text{Min}_{i \in I} a_i$; $m = 0$; $p = [\log_2 n]$; $x_0 = x_1 = \cdots = x_p = 0$; $\alpha_0 = \alpha_1 = \cdots = \alpha_p = 0$.

(ii) *Does job $m + 1$ belong to S?* If $a_{m+1} > t$, go to (iv); otherwise, go to (iii).

(iii) *Insertion of a job in S.* Calculate the lowest subscript of an empty set and set: $S_{k_0} = \{m\} \cup S_0 \cup \cdots \cup S_{k_0-1}$.

(iv) *Selection of job j.* If all the jobs are scheduled, go to (vi); if all the jobs of S are scheduled, set $t = a_{m+1}$ and go to (ii); otherwise, for every non-empty set S_k, the jobs of which are not all scheduled, we determine the job not already scheduled, j_k, with q_{j_k} maximal; j will be the job j_k with the greatest q_{j_k}.

(v) *Schedule of job j.* Set $t_j = t$, $t = t + d_j$ and go to (ii).

(vi) *Write the schedule.*

Proposition 2. *The complexity of the modified algorithm is* $O(n \log n)$.

Proof. (i) is a sort; the cost is $O(n \log n)$. Test (ii) is made no more than n times: the cost is $O(n)$. In (iii) every job is sorted no more than p times; the cost is $O(np) = O(n \log n)$. In (iv), we choose j by determining the minimum of $p + 1$ elements; the total cost is $O(n \log n)$. (v) and (vi) cost $O(n)$.

5. Branch and bound method

Our branch and bound method is based on the Schrage algorithm, on the critical set J and on the critical job c.

Description of the tree. We associate with every node S of the tree a one-machine problem and a lower bound $f(S)$. The upper bound f_0 is the value of the best solution known so far.

Branching. We consider the node of the tree with the lowest bound and apply the Schrage algorithm to the one-machine problem associated with it.

If c does not exist, the schedule is optimal (by the theorem); otherwise, the distance to the optimum is less than d_c, and either c will be processed before all the jobs of J, or c will be processed after all the jobs of J.

We consider two new problems (Fig. 4); in the first problem, we require job c to be processed before all jobs of J by setting

$$q_c = \text{Max}\left(q_c, \sum_{r \in J} d_r + q_p \right);$$

in the second problem, we suppose that job c is processed after all the jobs of J by setting

$$a_c = \text{Max}\left(a_c, \underset{r \in J}{\text{Min}}\, a_r + \sum_{r \in J} d_r \right).$$

The lower bound of the new nodes will be the maximum of $f(S)$, $h(J)$ and $h(J \cup \{c\})$; a new node will be added to the tree only if its lower bound is less than the upper bound f_0.

Upper bound. Every time we apply the Schrage algorithm, we compare the makespan to f_0. Moreover, we build another schedule by conserving the order of jobs except for job c that is processed after J.
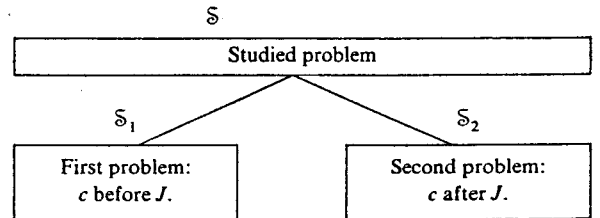

| S |
| Studied problem |

| S_1 | S_2 |
| First problem: c before J. | Second problem: c after J. |

Fig. 4. Branching.

Tests. The two tests below increase the efficiency of this method.

Let

$$K = \{ k/k \in I - J \text{ and } d_k > f_0 - h(J) \};$$

if $k \in K$, then job k has to be processed either before all jobs of J, or after all jobs of J.

Test 1. If $k \in K$ and $a_k + d_k + \Sigma_{r \in J} d_r + q_p > f_0$, then job k will be processed after all the jobs of J in an optimal schedule; so we set:

$$a_k = \text{Max}\left(a_k, \underset{r \in J}{\text{Min}} a_r + \sum_{r \in J} d_r \right).$$

Test 2. If $k \in K$ and

$$\underset{r \in J}{\text{Min}} a_r + d_k + \sum_{r \in J} d_r + q_k > f_0,$$

then job k will be processed before all the jobs of j in an optimal schedule; so we set:

$$q_k = \text{Max}\left(q_k, \underset{r \in J}{\text{Min}} q_r + \sum_{r \in J} d_r \right).$$

Treatment of the example. The makespan of the Schrage schedule is 53 (see Fig. 1); moreover $c = 1$ and $J = \{2, 3, 4\}$. If we process c before J, $h(J \cup \{1\}) = 53$; so, we suppose that c is processed after J by setting $a_1 = 28$.

We now apply the algorithm anew; the makespan of the schedule is now 50 (Fig. 5); we have
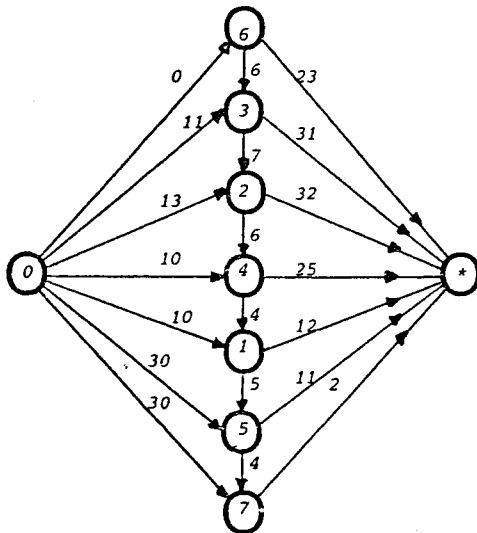

Fig. 5. Optimal schedule.

$c = 3$ and $J = \{2\}$. If c is before J the lower bound is 50; if c is after J the lower bound is 51; so Fig. 5 shows the optimal schedule.

6. Relaxed problem

Florian et al. [5], Lageweg et al. [8] also build Schrage schedules and define particular sets I_1 in order to calculate a lower bound. Baker and Su [2] allow preemption in order to bound the makespan; in other words a job can be preempted and continued later from the point in its processing when interruption occurred. This problem is solved by a preemptive algorithm; we apply the Schrage algorithm, but we stop the processing of job j when a job i with higher priority becomes available.

Proposition 3. *The makespan of the preemptive schedule is* $V = \text{Max}_{I_1 \subseteq I} h(I_1)$; *so this preemptive schedule is optimal.*

Proof. (1) $V \geq \text{Max}_{I_1 \subseteq I} h(I_1)$. This is true since $h(I_1)$ is also a lower bound on the makespan for the preemptive problem.

(2) $V \leq \text{Max}_{I_1 \subseteq I} h(I_1)$. The idea is to show that there exists I_0 so that $V = h(I_0)$. We replace each job i by d_i 'new jobs' with processing times equal to 1, release data a_i and tail q_i. We apply the Schrage algorithm to this new problem; the schedule is optimal since all the processing times are equal to 1. The makespan is equal to

$$V = \underset{r \in J}{\text{Min}} a_r + \sum_{r \in J} d_r + \underset{r \in J}{\text{Min}} q_r.$$

If a 'new job' r is the son of job i, then every son r' of i belongs to J; otherwise if $K = J \cup \{r'\}$,

$$h(K) = \underset{r \in K}{\text{Min}} a_r + \sum_{r \in K} d_r + \underset{r \in K}{\text{Min}} q_r = V + 1$$

would be a lower bound on the makespan, a contradiction. Let I_0 the set of jobs having a son in J; we have $h(I_0) = V$.

Proposition 3 illustrates the interest of using preemption to calculate a lower bound: preemption was allowed in order to improve our bound.

Proposition 4. *The number of preemptions of the preemptive algorithm is not greater than* $n - 1$.

Proof. There are no more than $2n - 1$ starts or restarts of jobs, since they take place either when a job is inserted in set S (no more than n times: a job is inserted once), or when a job is achieved (no more than $n - 1$ times: the machine is idle when the last job is achieved). The number of starts of jobs is equal to n, so there are no more than $n - 1$ restarts and preemptions.

Corollary 1. *The complexity of the preemptive algorithm is $O(n \log n)$.*

Proof. The complexity is $O((2n - 1) \log n) = O(n \log n)$ (Proposition 4).

Corollary 2. *The bound $n - 1$ can be reached.*

Proof. When the increasing order of a_i and the increasing order of q_i are two identical strict orders, and when $d_i > \text{Max}_{i \in I} a_i - \text{Min}_{i \in I} a_i$, there will be $n - 1$ preemptions.

Remark. The makespan of the preemptive schedule for the example is equal to 49; so 49 is a lower bound on the optimal makespan ($J = \{2, 3, 4\}$).

7. Results

The branch and bound method was coded in FORTRAN on an IRIS 80 and tested initially on 1000 problems. For each problem with n jobs, $3n$ integers were generated with uniform distributions between 1 and a_{max}, d_{max}, q_{max}.

We set $d_{max} = 50$, $a_{max} = q_{max} = \frac{1}{50} n d_{max} K$ and tried 50 values for K and 20 values for n: $K = 1, 2, 3, \ldots, 25$; $K = 30, 35, \ldots, 100$; $K = 110, 120, \ldots, 200$; $n = 50, 100, \ldots, 1000$. Of the 1000 problems, 999 were solved optimally. The problem with parameters $K = 19$ and $n = 850$ was not solved; the tree

was stopped with 111 nodes, the lower bound was equal to 29800, and the makespan of the best schedule was 29802 (the distance to the optimum was 0, 1 or 2). The hardest problems take place when $K = 18$, 19 and 20. In nine cases out of ten, either the Schrage solution, or the schedule when c is after J was optimal. In most examples, the cardinal of J is either equal to 1 (nearly every time when $K \geq 15$), or equal to n (nearly every time when $K \leq 12$). We also tested 12 instances with $n = 4000$, 7000, 10000 and $K = 10, 20, 30, 40$; each of these problems was solved optimally with only one node in the tree.

References

[1] K.R. Baker, *Introduction to Sequencing and Scheduling* (Wiley, New York, 1974).

[2] K.R. Baker and Z.S. Su, Sequencing with due dates and early start times to minimize tardiness, *Naval Res. Logist. Quart. 21* (1974) 171–176.

[3] P. Bratley, M. Florian and P. Robillard, On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/G/F_{max})$ problem, *Naval Res. Logist. Quart. 20* (1973) 57–67.

[4] J. Carlier, Problème à une machine, Rapport du groupe combinatoire de l'AFCET (1976).

[5] M. Florian, R. Trepant and G. Mac Mahon, An implicit enumeration algorithm for the machine sequencing problem, *Management Sci. 17*, 782–792.

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[7] 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 (1977).

[8] B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, Minimizing maximum lateness on one machine: computational experience and some applications, *Statistica Neerlandica 30* (1976) 25–41.

[9] B. Roy, *Algèbre Moderne et Théorie des Graphes* (Dunod, Paris, 1970).