# MINIMIZING TOTAL COSTS IN ONE-MACHINE SCHEDULING

A.H.G. Rinnooy Kan°, B.J. Lageweg*, J.K. Lenstra*

°Graduate School of Management, Delft, The Netherlands
*Mathematisch Centrum, Amsterdam, The Netherlands

## 1. INTRODUCTION

Suppose we have n *jobs* $J_1,\ldots,J_n$ that arrive simultaneously at time $t = 0$ to be processed on a continuously available *machine* which can handle only one job at a time. $J_i$ takes $p_i$ time units to be processed; costs $c_i(t)$, non-decreasing in t, are incurred if $J_i$ is completed at time t. We seek to find a schedule with associated completion times $t_i$ that minimizes $\sum_{i=1}^{n} c_i(t_i)$.

The problem is trivial in the case of *linear* cost functions $c_i(t) = \alpha_i(t-d_i)$, where $d_i$ can be interpreted as a due date for $J_i$; the optimal schedule has $J_j$ preceding $J_k$ if $p_j/\alpha_j \leq p_k/\alpha_k$. The *general* problem is considerably more difficult: a special case where $c_i(t) = 0$ if $t \leq d$, $c_i(t) = \alpha_i$ if $t > d$ belongs to Karp's list of *NP-complete* problems [8]. A polynomial-bounded algorithm for this problem would lead to efficient algorithms for a number of classic computational problems, and its existence seems highly unlikely.

*Dynamic programming* has been applied to solve the general problem [7,9], but most researchers have concentrated on the *weighted tardiness* function $c_i(t) = \alpha_i \max\{0,t-d_i\}$. This problem has been attacked by methods of implicit enumeration, notably of the *branch-and-bound* type [13], supported by *elimination criteria* that establish precedence relations between jobs [4,14]. Comparison of some of these methods [1] suggests that their rather poor performance may be due to the absence of strong lower bounds. We try to fill this gap by introducing an algorithm for the general problem, that performs satisfactorily in the special case of the weighted tardiness criterion. The algorithm and the computational experiments are described in more detail in [12].

*B. Roy (ed.), Combinatorial Programming: Methods and Applications,* 343–350. 

## 2. DESCRIPTION OF THE ALGORITHM

In sections 2.1, 2.2 and 2.3 an *enumeration scheme*, *elimination criteria* and a *lower bound* are proposed that together define our *branch-and-bound* algorithm. We include some remarks on the implementation of the elimination criteria and the lower bound.

### 2.1. Enumeration scheme

Since an optimal solution without machine idle time always exists [11], the total time needed to process a set of jobs is independent of the processing order.

We create a search tree as follows. From the root node, where no jobs have been scheduled, n branches lead to n nodes on the first level, each of which corresponds to a particular job being scheduled in the n-th position. Generally, each node in the tree corresponds to a set $\{J_i | i \in S' \subset \{1,\ldots,n\}\}$ filling the last $|S'|$ positions in a given order. By successively placing each job $J_r$ ($r \in S = \{1,\ldots,n\}-S'$) in the $|S|$-th position, $|S|$ new nodes are created. $J_r$ then runs from $P(S)-p_r$ to $P(S)$, where $P(Q) = \sum_{i \in Q} p_i$ for any $Q \subset \{1,\ldots,n\}$.

### 2.2. Elimination criteria

Successive application of the theorems in this section in each node of the tree will lead to sets $\{J_h | h \in B_i\}$ and $\{J_\ell | \ell \in A_i\}$ that respectively *precede* or *follow* $J_i$ in at least one optimal schedule. We can then limit the search to schedules satisfying these precedence constraints. In the following, implications for the weighted tardiness criterion will be presented as corollaries.

THEOREM 1. *At least one optimal solution has* $J_j$ *preceding* $J_k$ ($j,k \in S$) *if (a)* $c_j(t)-c_k(t)$ *is non-decreasing in* t *on the interval* $(P(B_k)+p_k, P(S-A_j))$, *and (b)* $p_j \leq p_k$.

*Proof.* If in any schedule $J_k$ precedes $J_j$, with $J_k$ starting at time C and $J_j$ finishing at time D, consider the schedule obtained by interchanging $J_k$ and $J_j$. The contribution to total costs by all jobs except $J_k$ does not increase, because of condition *(b)*. As to the joint contribution of $J_j$ and $J_k$, we have from condition *(a)* $c_j(D)-c_k(D) \geq c_j(C+p_k)-c_k(C+p_k)$, and from *(b)* $c_j(C+p_k) \geq c_j(C+p_j)$, together implying $c_j(D)+c_k(C+p_k) \geq c_j(C+p_j)+c_k(D)$. (Q.E.D.)

COROLLARY 1. *At least one optimal schedule has* $J_j$ *preceding* $J_k$ ($j,k \in S$) *if* $d_j \leq \max\{d_k, P(B_k)+p_k\}$, $\alpha_j \geq \alpha_k$, *and* $p_j \leq p_k$.

THEOREM 2. *At least one optimal schedule has* $J_j$ *preceding* $J_k$ ($j,k \in S$) *if (a)* $c_k(P(B_k)+p_k) = c_k(P(S-A_j)-p_k)$, *and (b)* $c_j(t)-c_k(t)$ *is non-decreasing in* t *on the interval* $(P(S-A_j)-p_k, P(S-A_j))$.

*Proof.* If $p_j \leq p_k$, we can apply Theorem 1. If $p_j > p_k$ and $J_k$ precedes $J_j$, with $J_k$ starting at C and $J_j$ finishing at D, consider the schedule obtained by putting $J_k$ directly after $J_j$. The contribution to total costs by all jobs except $J_k$ does not increase. As to the costs of $J_j$ and $J_k$, we have from *(a)* $c_k(D-p_k) = c_k(C+p_k)$, and from *(b)* $c_j(D)-c_k(D) \geq c_j(D-p_k)-c_k(D-p_k)$, together implying $c_j(D)+c_k(C+p_k) \geq c_j(D-p_k)+c_k(D)$. (Q.E.D.)

COROLLARY 2. *At least one optimal schedule has* $J_j$ *preceding* $J_k$ $(j,k \in S)$ *if* $d_k \geq P(S-A_j)-p_k$, $d_j \leq d_k$, *and* $\alpha_j \geq \alpha_k$.

THEOREM 3. *At least one optimal schedule has* $J_j$ *preceding* $J_k$ $(j,k \in S)$ *if* $c_k(P(B_k)+p_k) = c_k(P(S-A_j))$.

COROLLARY 3. *At least one optimal schedule has* $J_j$ *preceding* $J_k$ $(j,k \in S)$ *if* $d_k \geq P(S-A_j)$.

THEOREM 4. *In at least one optimal schedule* $J_k$ $(k \in S)$ *comes last among* $\{J_j | j \in S\}$ *if* $c_k(p_k) = c_k(P(S))$.

COROLLARY 4. *In at least one optimal schedule* $J_k$ $(k \in S)$ *comes last among* $\{J_j | j \in S\}$ *if* $d_k \geq P(S)$.

Theorems 3 and 4 are special cases of Theorem 2. Corollary 1 is given in [13]. Corollaries 1, 2 and 3 are extended versions of Theorems 1, 2 and 3 in [4]. Our proofs, however, are more general and considerably simpler than the original ones. Corollary 4 is also known as *Elmaghraby's Lemma* [3].

The only problem arising with the *implementation* of these elimination criteria is the possible creation of *precedence cycles* whenever two theorems seemingly contradict each other. These cycles are avoided by constructing the *transitive closure* of the set of known precedence relations immediately after a new relation has been found and by only examining pairs $(J_j,J_k)$ that are not yet related.

For a general cost function Theorems 1 to 4 can be applied in every node. In testing the algorithm on the weighted tardiness cost function, however, we applied Corollaries 1, 3 and 2 only in the root node, repeatedly running through them in the above order until no improvement was possible. Corollary 4 was checked in every node.

Precedence relations that are *a priori* given can be handled by the algorithm in an obvious way.

## 2.3. Lower bound

The lower bound LB on the costs of all possible schedules in a node has the form $LB = c(S')+LB^*$. Here $c(S')$ denotes the costs incurred by $\{J_i\}_{i \in S'}$, and $LB^*$ is a lower bound on the costs $c^*$ of an optimal schedule $\pi^* = (\pi^*(1),\ldots,\pi^*(m))$ for the jobs in $\{J_i\}_{i \in S}$, which are renumbered from 1 to m $(= |S|)$.

If all $p_i$ are equal, then the costs of putting $J_i$ in position $j$ are $c_{ij} = c_i(jp_1)$, and $\pi^*$ is obtained from the solution $(x^*_{ij})$ to the following *linear assignment problem* (cf. [9]):

$$\min\Big\{\sum_{i=1}^m \sum_{j=1}^m c_{ij}x_{ij} \;\Big|\; \sum_{j=1}^m x_{ij} = 1 \;\; (i = 1,\ldots,m),\; \sum_{i=1}^m x_{ij} = 1 \;\; (j = 1,\ldots,m),\; x_{ij} \geq 0 \;\; (i,j = 1,\ldots,m)\Big\}. \tag{1}$$

If not all $p_i$ are equal, this idea can be used to compute a lower bound, in two ways. Assuming all $p_i$ are integers, we can treat each job $J_i$ as $p_i/g$ new jobs, where $g = g.c.d.(p_1,\ldots,p_n)$, thus turning (1) into a $(P(S)/g)\times m$ *linear transportation problem*. However, it seems difficult to define effective cost coefficients for a general cost function (see [6] for a special case). Since job splitting can occur and the problem will often be large, we prefer a different approach. We define:

$$R_i(k) = \min_Q\{P(Q) \mid Q \subset \{1,\ldots,m\}-(B_i\cup\{i\}\cup A_i), |Q| = k\};$$

$$t_{ij} = P(B_i) + p_i + R_i(j-|B_i|-1);$$

$$c_{ij} = \begin{cases} c_i(t_{ij}) & \text{for } |B_i| < j \leq |\{1,\ldots,m\}-A_i|, \\ \infty & \text{otherwise.}\end{cases}$$

Solution of (1) now gives the desired lower bound $LB^*$, since $c_{ij}$ is a lower bound on the costs of putting $J_i$ in position $j$ and since $\pi^*$ is a feasible solution to (1):

$$c^* \geq \sum_{j=1}^m c_{\pi^*(j)j} \geq LB^*.$$

We now turn to the *implementation* of this lower bound. In any node, the solution to (1) can also be evaluated as a schedule, which may lead to a decrease in the value UB of the best schedule found so far. If $LB \geq UB$, the node can be eliminated. Otherwise, the jobs in $\{J_i \mid i \in S, S\cap A_i = \emptyset\}$ are candidates for position $m$. A complete solution of the assignment problems in these descendant nodes may be avoided by exploiting the solution $(x^*_{ij})$ to (1) and the soltuion $(u^*_i,v^*_j)$ to its dual:

$$\max\Big\{\sum_{i=1}^m u_i + \sum_{j=1}^m v_j \;\Big|\; u_i+v_j \leq c_{ij} \;\; (i,j = 1,\ldots,m)\Big\}.$$

Observing that $(u^*_i,v^*_j)_{i\neq r, j\neq m}$ is a feasible dual solution to the assignment problem, obtained from (1) by deleting row $r$ and column $m$, we see that a simple lower bound on the costs of scheduling $J_r$ in position $m$ is given by:

$$LB_r = c(S'\cup\{r\}) + (LB^*-u^*_r-v^*_m) = LB + (c_{rm}-u^*_r-v^*_m) \geq LB.$$

Branches for which $LB_r \geq UB$ can be pruned immediately. From the remaining candidates, a job $J_r$ with minimal $LB_r$ is scheduled in

position m. If application of elimination criteria does not further increase $LB_r$, we have to solve a new $(m-1)\times(m-1)$ assignment problem. Here we can still profit from $(x^*_{ij})$ and $(u^*_i,v^*_j)$, in two ways.

(a) The earliest possible finishing times $t_{ij}$ will not decrease, neither will the cost coefficients $c_{ij}$. So $(u^*_i,v^*_j)$ provides a *feasible dual* solution to the new problem.

(b) $(x^*_{ij})$ provides a *partial primal* solution to the new problem, that can be made *orthogonal* to the given dual solution by resetting $x^*_{ij} = 0$ if $u^*_i+v^*_j < c_{ij}$.

Remark (a) suggests an alternative bounding mechanism, whereby the assignment problem is solved only in the root node and provides bounds throughout the whole search tree by sums of appropriate dual variables (cf. [6]). Although we obtained reasonable computational results with this approach, we preferred the stronger bound; even then the trees may become quite large for moderate size problems.

In selecting a method for solving the assignment problems, ideally we would like to have a fast algorithm, not requiring an initial basic solution and producing a sequence of non-decreasing feasible dual solutions each of which may lead to early elimination of the current node. Dorhout's dual method [2] turned out to be more suitable than primal methods such as the stepping-stone algorithm or primal-dual ones such as the Hungarian method.

Dorhout's algorithm works on a complete bipartite graph $G = (S,T,E)$ where S and T correspond to unscheduled jobs and unfilled positions; edge $e_{ij} \in E$ has weight $w_{ij} = c_{ij}-u_i-v_j$. A partial primal solution $(x_{ij})$, orthogonal to a feasible dual solution $(u_i,v_j)$, defines a *matching* on G. The algorithm constructs the shortest augmenting path from any *exposed* vertex in S to the nearest exposed vertex in T, augments the matching and restores the orthogonality while the dual feasibility is maintained.

## 3. COMPUTATIONAL EXPERIMENTS

### 3.1. Tested algorithms

Our algorithm was tested on the weighted tardiness criterion and compared with Shwimer's algorithm [13] and a simple lexicographic method [10]. Both of them use the enumeration scheme described in section 2.1 and a lower bound corresponding to our $LB_r$.

Shwimer applies Corollary 4 and the static part of Corollary 1 (i.e. $d_j \le d_k$). His lower bound is given by:

$$LB'_r = c(S'\cup\{r\}) + \min_{i\in S-\{r\}}\{\alpha_i \max\{0,P(S-\{r\})-d_i\} + \min_{h\in S-\{r,i\}}\{\alpha_h\}\cdot T_{max}(S-\{r,i\})\},$$

where $T_{max}(Q)$ is the minimal maximal tardiness over all schedules of $\{J_h\}_{h\in Q}$, found by ordering the jobs according to increasing $d_h$ (see [11]). This bound can be computed very quickly, but depends explicitly on a property of the tardiness function.

The lexicographic method always chooses from the remaining candidates a job $J_r$ with maximal $d_r$, applies Corollary 4 and uses a lower bound $LB''_r = c(S' \cup \{r\})$.

### 3.2. Test problems

Each test problem with n jobs is specified by n integer triples $(p_i, d_i, \alpha_i)$. Their distribution is determined by four parameters: $\rho$ (correlation between processing times and due dates), s (relative variation of processing times), t (average tardiness factor), and r (relative range of due dates).

The $\alpha_i$ are generated from a uniform distribution over the interval (4.5,15.5). The $p_i$ are generated from a normal distribution with mean $\mu = 100$ and variance $s\mu$. If $\rho = 0$, the $d_i$ are generated from a uniform distribution with mean $(1-t)n\mu$ and variance $(rn\mu)^2/12$. If $\rho > 0$, each $d_i$ is generated in a similar way by replacing $\mu$ by $p_i$; this leads to $\rho = (1-t)/\sqrt{\{(1+1/s^2)r^2/12 + (1-t)^2\}}$.

The parameters $\rho$, t and r were found to influence the performance of other tardiness algorithms [1]; s was introduced because of its possible influence on our lower bound. We would expect *a priori* problems with positive $\rho$ (see [4, Corollary 1.3]), small s, very small or very large t (see [14]) and large r to be relatively easy for our method.

### 3.3. Computational results

We generated problems with 15 and 20 jobs, setting the parameters defined above to various values. The three algorithms were coded in ALGOL 60 and run on the CD 73-28 of the SARA Computing Centre in Amsterdam. The computational results can be found in Table 1.

The parameter t has a major influence on the performance of the algorithms, problems with t=.2 or t=.4 being "easy" and problems with t=.6 or t=.8 being "difficult". As to the other parameters, $\rho$ has no detectable influence, and problems with s=.05 and r=.95 are indeed easier than problems with s=.25 and r=.20.

On the easy problems, the lexicographic method runs quickly through large search trees; Shwimer's algorithm also performs well. Our algorithm creates very small trees, but it seems hardly worth while to compute sophisticated lower bounds for these problems.

On the difficult problems, however, our algorithm is by far superior to the other methods. Both of them fail on all twelve problems with 20 jobs; our method finishes seven of them, while the best solutions to the remaining five are better than Shwimer's.

Recently, Fisher [5] developed a dual average tardiness algorithm that uses a subgradient approach to produce strong lower bounds. Our algorithm performs very well on his test problems. However, they are easy ones with t=.5 and r=1, and both methods cannot be compared from these data alone.

## 4. CONCLUDING REMARKS

Our main conclusion has to be that, although our elimination criteria and our lower bound turn out to be useful, this one-machine problem remains a very difficult one.

An easy extension of our algorithm would be to check all elimination criteria anew in every node. More criteria might be found by considering the effects of moving three or more jobs at a time.

Our lower bound could be strengthened by explicitly respecting known precedence relations in solving the assignment problem. It is difficult to predict the effectiveness of this approach.

The idea of computing lower bounds by solving assignment problems whose coefficients $c_{ij}$ underestimate the costs of putting job $J_i$ in position j, can be applied to a wider set of problems, e.g. to minimizing total costs in an m-machine flow shop. This seems an interesting topic for future research.

We think that it would be worth while to develop very sharp bounds for the upper levels of the search tree and gradually simpler ones as we move down the tree and more extensive enumeration becomes attractive. Although our first experiments with such a *gliding lower bound* were disappointing, the idea could become useful in the future.

In spite of all the work done so far, the problem of minimizing total costs on one machine is likely to remain a challenge to researchers for a long time to come.

| number of jobs n | | 15 | | | | 20 | | | |
|---|---|---|---|---|---|---|---|---|---|
| tardiness t | | .2 | .4 | .6 | .8 | .2 | .4 | .6 | .8 |
| number of problems | | 12 | 12 | 12 | 12 | 6 | 6 | 6 | 6 |
| median solution time | Our Alg. | .0 | .8 | 6.3 | 45.6 | .8 | 1.1 | 180.8 | 300 * |
| | Shwimer | .0 | .6 | 76.7 | 300 * | .2 | 2.2 | 300 * | 300 * |
| | Lex.Alg. | .0 | .2 | 60 * | 60 * | .1 | 1.8 | 60 * | 60 * |
| maximum solution time | Our Alg. | .6 | 8.2 | 121.8 | 85.6 | 1.2 | 20.3 | (2)* | (3)* |
| | Shwimer | .3 | 3.9 | (3)* | (12)* | .3 | 10.2 | (6)* | (6)* |
| | Lex.Alg. | .3 | 14.8 | (10)* | (12)* | .2 | 21.6 | (6)* | (6)* |
| median number of nodes | Our Alg. | 1 | 44 | 647 | 4532 | 9 | 25 | 11105 | - |
| | Shwimer | 1 | 86 | 13066 | - | 12 | 281 | - | - |
| | Lex.Alg. | 1 | 305 | - | - | 105 | 3564 | - | - |
| maximum number of nodes | Our Alg. | 28 | 541 | 9564 | 9952 | 29 | 1206 | - | - |
| | Shwimer | 69 | 586 | - | - | 29 | 1130 | - | - |
| | Lex.Alg. | 572 | 36231 | - | - | 580 | 57671 | - | - |

Table 1. Computational results: solution time (in CPU seconds) and number of nodes (including eliminated nodes).
ℓ * : the median solution time exceeds the time limit ℓ.
(k)* : the time limit is exceeded k times.

ACKNOWLEDGEMENT

B. Dorhout's cooperation in making available his assignment code is gratefully acknowledged.

REFERENCES

1. K.R. BAKER, J.B. MARTIN, An Experimental Comparison of Solution Algorithms for the Single-Machine Tardiness Problem, *Nav.Res. Log.Quart.* 21(1974)187-199.
2. B. DORHOUT, Experiments with Some Algorithms for the Linear Assignment Problem, Report BW 39, Mathematisch Centrum, Amsterdam, 1974.
3. S.E. ELMAGHRABY, The One-Machine Sequencing Problem with Delay Costs, *J.Ind.Eng.* 19(1968)105-108.
4. H. EMMONS, One-Machine Sequencing to Minimize Certain Functions of Job Tardiness, *Opns.Res.* 17(1969)701-715.
5. M.L. FISHER, A Dual Algorithm for the One-Machine Scheduling Problem, Report 7403, Graduate School of Business, University of Chicago, 1974.
6. L. GELDERS, P.R. KLEINDORFER, Coordinating Aggregate and Detailed Scheduling Decisions in the One-Machine Job Shop: Part I. Theory, *Opns.Res.* 22(1974)46-60.
7. M. HELD, R.M. KARP, A Dynamic Programming Approach to Sequencing Problems, *J.SIAM* 10(1962)196-210.
8. R.M. KARP, Reducibility among Combinatorial Problems, pp.85-103 in R.E. MILLER, J.W. THATCHER (eds.), *Complexity of Computer Computations*, Plenum Press, New York-London, 1972.
9. E.L. LAWLER, On Scheduling Problems with Deferral Costs, *Man. Sci.* 11(1964)280-288.
10. J.K. LENSTRA, Recursive Algorithms for Enumerating Subsets, Lattice-Points, Combinations and Permutations, Report BW 28, Mathematisch Centrum, Amsterdam, 1973.
11. A.H.G. RINNOOY KAN, The Machine Scheduling Problem, Report BW 27, Mathematisch Centrum, Amsterdam, 1973; Report R/73/4, Graduate School of Management, Delft, 1973.
12. A.H.G. RINNOOY KAN, B.J. LAGEWEG, J.K. LENSTRA, Minimizing Total Costs in One-Machine Scheduling, *Opns.Res.* (to appear).
13. J. SHWIMER, On the *N*-Job, One-Machine, Sequence-Independent Scheduling Problem with Tardiness Penalties: a Branch-and-Bound Solution, *Man. Sci.* 18(1972)B301-313.
14. V. SRINIVASAN, A Hybrid Algorithm for the One-Machine Sequencing Problem to Minimize Total Tardiness, *Nav.Res.Log.Quart.* 18(1971)317-327.