Theory and Methodology

Minimum loss scheduling problems

W.M. Nawijn

Faculty of Applied Mathematics, University of Twente, Enschede, Netherlands

Received October 1989; revised April 1990

Abstract: We consider two minimum loss scheduling problems. Let the arrival epochs and processing times of n consecutive jobs be known. The first problem deals with the case these jobs can be processed by m parallel nonidentical machines without waiting room, i.e., a pure loss system. In the second problem there is one machine with a finite waiting room, i.e., a single machine delay-loss system. For both problems we give an algorithm that minimizes the total number (c.q. weight) of lost jobs.

Keywords: Scheduling, network

Introduction

Let the subsequent arrival epochs t_j ($1 \leq j \leq n$) of n future jobs be known. In the first scheduling problem we consider, these jobs can be processed on m nonidentical parallel machines. These machines constitute a loss system, i.e., there is no storage room at the machines to accommodate waiting jobs. Consequently, if all machines are busy upon arrival of a job, the job is lost. The problem is to find a schedule, i.e., an assignment of jobs to machines, for which the number or total weight of lost jobs is minimized, under the additional assumption of known processing times. This problem has been solved by Arkin and Silverberg [1] in case of identical machines. In the second scheduling problem we consider the jobs arrive at a single machine delay-loss system in which there is a finite buffer to store waiting jobs. Again the object is to find a minimum loss schedule. We restrict ourselves to the case that the processing sequence of jobs satisfies the first-come first-served discipline.

Problems of the above type have obtained little attention in the literature. In practice these problems arise in situations with scheduled arrivals of jobs with known service requirements, to be serviced at stations with bounded storage capacities, where one wants to prevent overflows to auxiliary stations or buffers, a phenomenon well known in material handling systems. A somewhat related so-called ‘minimum spillage sequencing’ problem has been considered by Tovey et al. [4].

1. Pure loss system

Let the set of jobs $= \{J_1, \dots, J_n\}$ be characterized as

$$\{(t_i, p_{ir}, w_{ir}), r \in S_i, i = 1, 2, \dots, n\}$$

in which $S_i \subseteq \{1, 2, \dots, m\}$ denotes the subset of machines that can process job J_i and in which w_{ir} is the weight or value of processing job J_i on machine M_r with p_{ir} the corresponding processing time. The case $S_1 = S_2 = \dots = S_n = \{1, 2, \dots, m\}$ will be referred to as the unrestricted machine case.

Problem P1. Find a feasible schedule (i.e., an assignment of jobs to machines) that maximizes the total weight of processed jobs.

Let us first consider the following two special problems for the unrestricted machine case.

Problem P2: P1 with $p_{ir} = p_i$ and $w_{ir} = 1$ ($1 \leq r \leq m$, $1 \leq i \leq n$).

Problem P3: P1 with $p_{ir} = p_i$ and $w_{ir} = w_i$ ($1 \leq r \leq m$, $1 \leq i \leq n$).

Consider an arbitrary machine M_r and suppose job J_i is processed on M_r . Then its processing interval is $I_i = [t_i, t_i + p_i)$. Two jobs J_i and J_j ($j \neq i$) cannot both be processed on M_r if $I_i \cap I_j \neq \emptyset$ (empty set). Let G be the intersection graph of the intervals I_i , $i = 1, 2, \dots, n$. By definition G is an interval graph. Since a job can be processed on one machine only and machines are identical we have the following result:

Proposition 1. Problem P2 (P3) can be solved as a maximum ($-weight$) m -coloring of the interval graph G .

Problem P2 is equivalent to a maximum m -covering problem for the complementary graph $\bar{G}$ of G . Since $\bar{G}$ is a comparability graph, which is transitive orientable, this problem can be polynomially solved in $O(mn^2)$ time, see e.g. Gavril [3]. For the more general problem P3 Arkin and Silverberg [1] have constructed an algorithm that solves it in $O(n^2 \log n)$ time for any m . They also prove that a subproblem of P1 with an *arbitrary* number of *identical* machines and an *arbitrary* number of *equally weighted* jobs, but *different* sets S_i , is NP-hard.

In the next section we will show that the general problem P1 for an *arbitrary* number of jobs but a *fixed* number of machines is polynomially solvable in $O(n^{m+1})$ time, generalizing the result in [1] to *nonidentical* machines

1.1. The general problem P1: a network model

Let $\{t_0, t_1, \dots, t_n\}$ be the ordered set of subsequent arrival epochs of the $n + 1$ jobs $J_0, J_1, \dots, J_n$, respectively, in which $t_0 (< t_1)$ is the arrival epoch of a fictitious job J_0 , which can be processed on every machine, with $p_{0r} = 0$ and, $r = 1, 2, \dots, m$. For every machine M_r and all jobs J_i that can be processed on it we define a simple digraph $G_r = (V_r, E_r)$ in which a vertex $i_r \in V_r$ corresponds to the interval $I_i = [t_i, t_i + p_{ir})$ for $i > 0$ and $I_0 = \{t_0\}$. The arc set E_r contains the arc (i, j) if and only if $i < j$ and $I_i \cap I_j = \emptyset$. The latter condition means that two jobs J_i and J_j that can be processed on machine M_r can both be scheduled on M_r . The graph G_r is a comparability graph. Notice that every feasible schedule on M_r corresponds to a path in G_r .

Now suppose a feasible schedule on the m machines is given. The state of the system at some arbitrary instant t can be represented by an m -vector $x(t) = (x_1(t), \dots, x_m(t))$ where $x_r(t) = i$, $r = 1, 2, \dots, m$, if at time t job J_i , $i = 1, 2, \dots, n$, is either being processed on M_r or, in case M_r is idle, it is the last job processed on M_r and where $x_r(t) = 0$ if until time t no job has been processed on M_r . Observe that state transitions occur at processing starts and that $x(t)$ is component wise nondecreasing. If at time t a state transition occurs, $x(t)$ changes in one component only. (If two or more arrival epochs coincide they can be perturbed to guarantee this fact.) In particular, let $x_r(t_-) \rightarrow x_r(t)$ be such a change. Then it should satisfy the conditions

(A1) $(x_r(t_-), x_r(t)) \in E_r$,

(A2) $x_r(t) > \max_{1 \leq r \leq m} x_r(t_-)$.

The first of these is a feasibility condition and the second one stems from the job ordering. Clearly, every feasible schedule can be represented as a sequence of consecutive state vectors satisfying the conditions (A1) and (A2) starting in state $(0, 0, \dots, 0)$.

In constructing the network model we form the Cartesian sum, cf. Berge [2], $H = G_1 + G_2 \cdots + G_m$ with vertex set $X = V_1 \times V_2 \times \cdots \times V_m$ and arc set A , defined such that the successors of $(x_1, \dots, x_m) \in X$ comprise the set

$$I(x_1, \dots, x_m) = \bigcup_{1 \leq r \leq m} \{(x_1, \dots, x_{r-1}, y, x_{r+1}, \dots, x_m) \mid (x_r, y) \in E_r\}.$$

In view of condition (A2), any feasible schedule can be represented in H as a path $\pi \subset A$ starting in vertex $(0, \dots, 0)$ such that for all arcs $(x, y) \in \pi$ we have $\max_r x_r < \max_r y_r$. Hence we delete from A all arcs $(x, y) \in A$ such that $\max_r x_r = \max_r y_r$. Subsequently, we add to H a terminal node T whose predecessors are the vertices

$$\{x \in X \mid (x_r, y) \notin E_r, \quad y > \max_{1 \leq i \leq m} x_i \text{ for all } r = 1, 2, \dots, m\}.$$

Let the resulting digraph be $D = (\bar{X}, \bar{A})$. Then every path from $S = (0, \dots, 0)$ to T in D corresponds to a feasible schedule and, conversely, every feasible schedule corresponds to a path from S to T . Finally, to each arc $(x, y) \in \bar{A}$, $y \neq T$, for which $x_r \neq y_r$, we adjoin a 'length' (c.q. weight) $w_{y_r, r}$ corresponding to the revenue obtained by processing job J_{y_r} on machine M_r ; arcs $(x, T) \in \bar{A}$ get length zero.

Clearly, the objective of finding a feasible schedule of maximum total weight is equivalent to finding a longest path from S to T in the network $N = (D, w)$, which is acyclic by construction.

Since in general $|S_r| = O(n)$, D will have $|\bar{X}| = O(n^m)$ vertices. To count the number of arcs observe that at most $m \times n$ arcs leave any vertex, therefore, the number of arcs $|\bar{A}| = O(n^{m+1})$. Since solving a longest path problem in an acyclic network with α arcs takes $O(\alpha)$ time, we have the following result:

Theorem 1. *For a fixed number of non-identical machines, m say, and an arbitrary number of jobs, n say, (with or without machine restrictions), the scheduling problem can be solved in $O(n^{m+1})$ time.*

This result generalizes a result for the case of identical machines with machine restrictions obtained by Arkin and Silverberg [1]. Observe that the weights (c.q. arc lengths) may also depend on the preceding job.

2. Delay-loss systems

Consider one machine ($m = 1$) with a buffer of capacity $b = 1$. We want to find a schedule with as many processed jobs as possible within the class of FCFS-schedules, i.e., such that the processing sequence is a subsequence of the arrival sequence.

To characterize a feasible schedule let $(j_1, j_2, \dots, j_i)$ be a (sub)sequence of jobs. Let s_{j_r} ($r = 1, 2, \dots, i$) denote the starting time of job j_r , then

$$s_{j_r} = \max(t_{j_r}, s_{j_{r-1}} + p_{j_{r-1}}), \quad r = 2, 3, \dots, i, \quad (1a)$$

$$s_{j_1} = t_{j_1}. \quad (1b)$$

For a buffer capacity one ($b = 1$) the subsequence corresponds to a feasible schedule if

$$s_{j_{r-1}} < t_{j_r} \quad (r = 2, 3, \dots, i), \quad (2)$$

since each job in the schedule must arrive after its predecessor's starting time.

Let π_{j_l} denote a subschedule of length l corresponding to the first l jobs of a feasible schedule with job j in position l . By an augmented schedule $(\pi_{j_l}, i_1, i_2, \dots, i_m)$ we mean the subschedule π_{j_l} augmented with the jobs $i_1, i_2, \dots, i_m$ in the given order.

Lemma. Given π_{jl}^1 and π_{jl}^2 , such that $s_j^1 < s_j^2$, then for any feasible schedule $(\pi_{jl}^2, i_1, i_2, \dots, i_m)$ the schedule $(\pi_{jl}^1, i_1, i_2, \dots, i_m)$ is feasible, too.

Proof. The assertion immediately follows from (1) and (2) by noting that the s_{j_r} ($r = 2, \dots, i$) are nondecreasing functions of s_{j_i} , while the numbers t_{j_r} ($r = 2, \dots, i$) are fixed. $\square$

For the two (sub)schedules in the Lemma we say that π_{jl}^1 dominates π_{jl}^2 , for if $(\pi_{jl}^2, i_1, i_2, \dots, i_m)$ is optimal, so is $(\pi_{jl}^1, i_1, i_2, \dots, i_m)$.

Definition. A dominating (sub)schedule $\bar{\pi}_{jk}$ of length k with terminal job j is a feasible (sub)schedule that dominates all feasible-schedules of length k with terminal job j .

Thus a dominating (sub)schedule $\bar{\pi}_{jk}$ is a feasible (sub)schedule π_{jk} for which job j starts as soon as possible. Of course, several such dominating schedules may exist, from which we need only one. Suppose that a schedule π of maximum length corresponds to the sequence of jobs $(j_1, j_2, \dots, j_m)$. Then we may assume that every (sub)schedule $\pi_{j_i, i}$, $i = 1, 2, \dots, m$, is a dominating (sub)schedule, since if for some $i = k$ $\pi_{j_k, k}$ is not, the schedule π can be changed into a feasible schedule $(\bar{\pi}_{j_k, k}, j_{k+1}, j_{k+2}, \dots, j_m)$ in which job j_k starts as soon as possible, without violating optimality.

The optimal schedule can thus be found by searching over the set of dominating schedules of all possible lengths with fixed terminal nodes. Let A_l be the set of jobs that can be scheduled as the l -th job in a feasible schedule. Moreover, let P_{jl} be the set of all possible predecessor jobs of job j in a feasible schedule of length l with job j as terminal job. Then, by definition,

$$A_1 = J = \{1, 2, \dots, n\}, \quad (3a)$$

$$A_l = \{j \in J \mid P_{jl} \neq \emptyset\} \quad \text{for } 2 \leq l \leq n, \quad (3b)$$

and

$$P_{j1} = \{0\}, \quad j \in J, \quad (4a)$$

$$P_{jl} = \{i \in J \mid i \in A_{l-1}, 1 \leq i \leq j-1, s_i < t_j\} \quad \text{for } 2 \leq l \leq j \leq n. \quad (4b)$$

For a schedule $\bar{\pi}_{jl}$ (if it exists) we introduce the pair (k_{jl}, s_{jl}) , $1 \leq l \leq j$, in which $k_{jl} \in P_{jl}$ denotes the predecessor node of j in $\bar{\pi}_{jl}$ and s_{jl} the corresponding starting time of job j .

Observe that

$$(k_{j1}, s_{j1}) = (0, t_j) \quad \text{for } 1 \leq j \leq n,$$

where $k_{j1} = 0$ denotes a dummy job.

Now suppose A_{l-1} and the pairs $\{(k_{i,l-1}, s_{i,l-1}) \mid i \in A_{l-1}\}$ are known. The next step is to determine A_l and the pairs $\{(k_{jl}, s_{jl})\}$ for all $j \in A_l$. By definition, cf. (3) and (4),

$$P_{jl} = \{i \in J \mid i \in A_{l-1}, 1 \leq i \leq j-1, s_{i,l-1} < t_j\}$$

and

$$A_l = \{j \in J \mid P_{jl} \neq \emptyset\}.$$

For each job $j \in A_l$ determine a node $r \in P_{jl}$ (i.e., the predecessor job of j in $\bar{\pi}_{jl}$, if it exist) such that

$$s_r + P_r = \min_{i \in P_{jl}} \{s_{i,l-1} + p_i\}. \quad (5)$$

In case of a tie choose a node arbitrarily.

Now the pair (k_{jl}, s_{jl}) follows from

$$k_{jl} = r, \quad (6a)$$

$$s_{jl} = \max(t_j, s_r + p_r). \quad (6b)$$

This procedure can be repeated until either a length L is found such that $L \leq n - 1$ with $A_{L+1} = \emptyset$, or, until a feasible schedule of length n is found. In the first case L is the length of the longest feasible schedule with in the second case all jobs can be scheduled. Having found $L \leq n - 1$, the jobs in the optimal schedule can be found by backtracking from the terminal node in using the pairs $\{(k_{jl}, s_{jl}) \mid j \in A_l, 2 \leq l \leq L\}$.

Theorem 2. *The algorithm generates the longest feasible schedule in $O(n^3)$ time.*

Proof. As indicated earlier the restriction to dominant subschedules is sufficient. Since the algorithm considers every pair (l, j) , $1 \leq l \leq j \leq n$, a dominating schedule $\bar{\pi}_{jl}$, if it exists, will always be found. Moreover, since it is impossible to have $A_l = \emptyset$ and $A_k \neq \emptyset$ such that $k > l$, if the algorithm stops for $l = L$, then L is the length of an optimal schedule. The number of pairs (l, j) to be considered is $O(n^2)$. For each pair (l, j) one has to determine P_{jl} , requiring $O(n)$ comparisons, and the pair (k_{jl}, s_{jl}) , which requires $O(\log n)$ time to obtain the minimum of at most n numbers. Therefore, the time complexity of the algorithm is $O(n^3)$. $\square$

Now suppose the buffer capacity for waiting jobs equals $b \geq 2$. In this case a schedule $(j_1, j_2, \dots, j_i)$, $i \geq b + 1$, is feasible if the jobs j_r satisfy

$$s_{j_{r-b}} < t_{j_r} \quad \text{for } b + 1 \leq r \leq i. \quad (7)$$

The above procedure should be generalized accordingly. Let $\bar{\pi}_{jl}$ be a dominating schedule. To indicate the successive jobs in this schedule we use the notation $\bar{\pi}_{jl} = (0, \bar{\pi}_{jl}(1), \dots, \bar{\pi}_{jl}(l))$, i.e., $\bar{\pi}_{jl}(l) = j$. The predecessor sets P_{jl} are defined by

$$P_{j1} = \{0\} \quad \text{for } 1 \leq j \leq n, \quad (8a)$$

$$P_{jl} = \emptyset \quad \text{for } 2 \leq l \leq n, 1 \leq j \leq l - 1, \quad (8b)$$

$$P_{jl} = \{l - 1, l, \dots, j - 1\} \quad \text{for } 2 \leq l \leq b, l \leq j \leq n, \quad (8c)$$

$$P_{jl} = \{i \in J \mid i \in A_{l-1}, 1 \leq i \leq j - 1, s_{r, l-b} < t_{j_r}, r = \bar{\pi}_{i, l-1}(l - b)\} \quad (8d)$$

for $b + 1 \leq l \leq j$, in which

$$A_l = \{l, l + 1, \dots, n\} \quad \text{for } 1 \leq l \leq b.$$

In determining P_{jl} for each job $i \in A_{l-1}$ one should check if job $\bar{\pi}_{i, l-1}(l - b) \in A_{l-b}$ satisfies feasibility condition (7) by backtracking along the jobs of $\bar{\pi}_{i, l-1}$. This requires nb operations at most. So the complexity in this case is $O(n^3b)$. Since the optimization problem is nontrivial only if $b < n - 1$, we conclude that the problem for arbitrary n and b is polynomial. Summarizing, we proved the following (formal) result:

Theorem 3. *Given n pairs $\{(t_j, p_j) \mid j = 1, 2, \dots, n\}$ such that $t_j \leq t_{j+1}$ and $p_j \geq 0$, and a number b ($1 \leq b < n - 1$), then the largest subset $\{j_1, j_2, \dots, j_i\} \subseteq \{1, 2, \dots, n\}$, such that $j_1 < j_2 < \dots < j_i$, satisfying (1) and (7) can be found in $O(n^4)$ time.*

Remark. Observe that the method solves the problem by a forward recursion and is easily implemented in tabular form in which the entry (l, j) contains the pair (k_{jl}, s_{jl}) and in which column j is characterized by p_j and t_j . Starting with row $l = 1$ the entries in the successive rows can be determined until a row is found for which no feasible entries can be found.

It is not difficult to see that in case all jobs have identical processing times the solution of the problem at hand is found almost trivially by applying the rule: accept a job if on its arrival the buffer is not full. The solution can be obtained in $O(n)$ time for arbitrary b .

Suppose to each job j an integer weight w_j is assigned and we want to find a schedule that maximize the total weight of scheduled jobs. The procedure for identical weights is easily adapted to this case. First it should be observed that the dominance property still holds, i.e., for a given attainable weight w a subschedule $\pi_{w_j}^{(1)}$, with total weight w and terminating with job j , for which job j starts at time s_j^1 , dominates a subschedule $\pi_{w_j}^{(2)}$ for which job j starts at time $s_j^2 > s_j^1$. Now the pair $(k_{w_j}, s_{w_j}^1)$ should be determined for each (feasible) weight-job combination (w, j) . For each such combination the possible predecessor jobs in the dominating subschedule $\bar{\pi}_{w_j}$ are found in $A_{w-w_j} \cap \{1, 2, \dots, j-1\}$. Let i be such a job. By backtracking along $\bar{\pi}_{i, w-w_j}$ until job $b-1$ before the last one is reached, one should again check if the resulting job satisfies feasibility condition (7); if so, $i \in P_{jw}$, otherwise, the combination (w, j) is infeasible. Since the number of weight-job combinations is $O(n \sum_1^n w_j)$, the solution can be found in $O(n^2 b \sum_1^n w_j)$ time

References

- [1] Arkin, E.M., and Silverberg, E.B., "Scheduling jobs with fixed start and end times", *Discrete Applied Mathematics* 18 (1987) 1–8.
- [2] Berge, C., *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.
- [3] Gavril, F., "Algorithms for maximum k -coloring and k -covering of transitive graphs", *Networks* 17 (1987) 465–470.
- [4] Tovey, C.A., Weiss, G., and Wilson, J.R., "Minimum spillage sequencing", *Management Science* 34 (1988) 306–330.