European Journal of Operational Research 78 (1994) 120–127
North-Holland

Theory and Methodology

Scheduling jobs with release times on a machine with finite storage

W.M. Nawijn, W. Kern and S. Baas

Faculty of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands

Received November 1991; revised September 1992

Abstract: Consider a single machine with a buffer which can store up to b waiting jobs for some fixed b. Given the release times, the weights and the processing times of n consecutive jobs, a maximum weight subset of jobs is to be found that is schedulable without violating the buffer's capacity constraint. A polynomial algorithm for the unweighted loss-delay problem is presented. The weighted case is shown to be NP-hard as well as an unweighted two-machine version.

Keywords: Scheduling; Delay-loss system; Complexity

1. Introduction

We consider a deterministic one-machine scheduling problem which differs from the classical one-machine sequencing problems in two respects. First we assume that there is only a finite buffer capacity $b \in \mathbb{Z}_+$ to store up to b waiting jobs. Secondly, the set of jobs J is characterized by a set of known triples $J = \{(r(i), p(i), w(i)), 1 \leq i \leq n\}$, where $r(i)$ is the release (c.q. arrival) time of job i, $p(i)$ its processing time and $w(i)$ its (nonnegative) weight. The jobs are ordered such that $0 \leq r(1) \leq r(2) \leq \cdots \leq r(n)$; jobs arriving at the same moment are ordered according to nondecreasing processing times. Any time a job arrives, it either has to be processed or put in the buffer, otherwise we assume it is lost. Our problem is to find a maximum weight subset of the jobs that can be processed without violating the buffer's capacity constraint at any time. If all weights are equal the problem amounts to finding a minimum loss schedule. The latter problem has been considered for identical parallel machines without buffers (a pure loss system) by Arkin and Silverberg [1]. In [4] we have shown that our problem for arbitrary b can be solved in $O(n^4)$ time if the processing order is restricted to first come–first served. It is assumed throughout that the system is empty initially.

In Section 2 we will present a polynomial time algorithm to find a minimum loss schedule (unweighted case) for fixed $b \in \mathbb{Z}_+$. In Section 3 we prove two complexity results. The first result concerns the weighted one-machine case for $b = 1$, which is proved to be NP-hard. The second result deals with a system of two parallel identical machines, either having a separate buffer of capacity $b = 1$ or sharing a common buffer of capacity $b = 2$. It will be shown that this problem is NP-hard, even in the unweighted case.

Correspondence to: W.M. Nawijn, Faculty of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands.

SSDI 0377-2217(93)E0133-I

2. A polynomial algorithm for the unweighted case

We first consider the minimum loss problem in the case that the buffer capacity equals one. Thus let $J = \{(r(i), p(i), 1), 1 \le i \le n\}$ be a set of (unweighted) jobs with release times $0 \le r(1) \le r(2) \le \cdots \le r(n)$ and processing times $p(i) > 0$. A feasible schedule may be described by specifying a subset of the jobs, say $\{i_1, \ldots, i_k\}$ and their starting times $s(i_1) < \cdots < s(i_k)$. The feasibility conditions are:

(i) $s(i_j) \ge \max(r(i_j), s(i_{j-1}) + p(i_{j-1}))$, $j = 2, 3, \ldots, k$; $s(i_1) = r(i_1)$;

(ii) the waiting intervals $[r(i_j), s(i_j)]$, $j = 1, 2, \ldots, k$, are mutually nonoverlapping.

Our problem is to find a feasible schedule of maximum 'size' k (= number of jobs processed).

We start out with some remarks. First notice that in condition (i) above we may as well replace the inequality by an equality. (There is no reason for taking a job from the buffer and not processing it immediately.) From this it is clear that a feasible schedule is completely characterized by specifying the sequence $\pi = (i_1, \ldots, i_k)$ of jobs, indicating the order in which they are processed. The starting times of the jobs which can be derived from π will be denoted by $s_\pi(i_1), \ldots, s_\pi(i_k)$, or simply by $s(i_1), \ldots, s(i_k)$. Any sequence π as above will be called a *feasible sequence*. Thus feasible sequences uniquely imply the starting times and buffer contents in a feasible schedule.

To describe a partial schedule, we need not only to specify the corresponding initial segment, say $(i_1, \ldots, i_\ell)$, of the feasible sequence π, but also to indicate whether there is some job i waiting in the buffer to be processed after job i_ℓ. To avoid ambiguities, we will indicate the buffer content-provided this is different from i_ℓ - at time $s(i_\ell) - \varepsilon$, where $\varepsilon > 0$ is small enough (smaller than any possible difference between starting times and release times). We therefore define a partial sequence of length ℓ to be a sequence π_ℓ of one of the following two types (assuming the obvious feasibility conditions to be satisfied):

Type I : $\pi_\ell = (i_1, \ldots, i_\ell;\ \text{buffer} = i)$ with $r(i) < r(i_\ell)$;

Type II: $\pi_\ell = (i_1, \ldots, i_\ell;\ \text{buffer} = \Phi)$.

Proposition 1. (i) *Any partial sequence uniquely describes a partial schedule.*

(ii) *In a partial schedule described by a type I sequence, job i_ℓ starts upon its release time, i.e. $s(i_\ell) = r(i_\ell)$.*

(iii) *Any partial schedule is described by a partial sequence.*

Proof. (i): Obviously, as in the case of (complete) feasible solutions, a partial sequence implies all information about the partial schedule up to the time where job i_ℓ is started. Note that it does not provide information about what happens at time $t \ge s(i_\ell)$. Thus, for example, it may well happen that in a schedule described by a type II sequence, a job is put into the buffer at time $s(i_\ell)$. The starting times of jobs $i_1, \ldots, i_\ell$, however, are easily deduced from the sequence. We will denote them by $s_{\pi_\ell}(i_1), \ldots, s_{\pi_\ell}(i_\ell)$ or simply $s(i_1), \ldots, s(i_\ell)$.

(ii): Note that, by definition, in a partial schedule described by a type I partial sequence, job i resides in the buffer at time $s(i_\ell) - \varepsilon$. Thus job i_ℓ cannot have been kept waiting in the buffer (note that the type I condition $r(i) < r(i_\ell)$ implies $i \ne i_\ell$), and therefore must have been started upon its release time, i.e. $s(i_\ell) = r(i_\ell)$.

(iii): Consider a partial schedule with jobs $i_1, \ldots, i_\ell$ being processed as the ℓ first jobs. There are three different cases:

a) The buffer is empty at time $s(i_\ell) - \varepsilon$. In this case the schedule is described by a type II sequence.

b) The buffer contains job i_ℓ at time $s(i_\ell) - \varepsilon$. In this case, the schedule also corresponds to a type II sequence.

c) The buffer contains some job $i \ne i_\ell$ at time $s(i_\ell) - \varepsilon$. In this case, the schedule is described by a type I sequence. Note that since job i is in the buffer at time $s(i_\ell) - \varepsilon$, we have $r(i) < s(i_\ell)$. As we have seen in (ii) above, $s(i_\ell) = r(i_\ell)$ must hold, hence the type I condition $r(i) < r(i_\ell)$ is fulfilled. □

Our algorithm for solving the minimum loss problem roughly consists of building up an optimal sequence step by step, successively increasing the *size* ℓ of a partial sequence π_ℓ. For this purpose, let us define the following sets of partial sequences of type I and II, respectively:

$$\Pi_\ell^{\mathrm{I}}(j) := \{(i_1, \ldots, i_\ell; \text{buffer} = i) \mid i_\ell = j\},$$

$$\Pi_\ell^{\mathrm{II}}(j) := \{(i_1, \ldots, i_\ell; \text{buffer} = \Phi) \mid i_\ell = j\}.$$

Thus $\Pi_\ell^{\mathrm{I}}(j)$ and $\Pi_\ell^{\mathrm{II}}(j)$ are all partial sequences of size ℓ of type I and II, respectively, in which job j is the last job to be processed.

As we will see below, we may restrict our attention to a single element in each of these sets. The reason is that within each such set, there is a sequence 'dominating' all others in a natural way.

Definition (Type I dominance). Let

$$\pi'_\ell = (i'_1, \ldots, i'_{\ell-1}, j; \text{buffer} = i') \text{ and } \pi''_\ell = (i''_1, \ldots, i''_{\ell-1}, j; \text{buffer} = i'')$$

be elements of $\Pi_\ell^{\mathrm{I}}(j)$. Then π'_ℓ *dominates* π''_ℓ if $p(i') \leq p(i'')$.

By $\pi_\ell^{\mathrm{I}}(j)$ we denote a sequence which dominates all other sequences in $\Pi_\ell^{\mathrm{I}}(j)$.

Proposition 2. *Let $\pi''_\ell \in \Pi_\ell^{\mathrm{I}}(j)$ correspond to an initial segment of an optimal sequence and let π'_ℓ dominate π''_ℓ. Then π'_ℓ corresponds to an initial segment of an optimal sequence as well.*

Proof. Suppose $\pi''_\ell \sigma'' i'' \rho''$ is an optimal sequence. We show that $\pi'_\ell \sigma'' i' \rho''$ is optimal, too. Since both sequences are of the same size it suffices to prove that the latter one is feasible. Recall that, since both π'_ℓ and π''_ℓ are of type I, the starting time of job j equals its release time in both sequences. Moreover, since i'' resides in the buffer during the processing of job j, all jobs in σ'' must be released on or after time $r(j) + p(j)$ and start processing upon release. So neither one of the jobs in π'_ℓ nor i' can occur in σ''. Having job i' instead of i'' in the buffer makes no difference, so $\pi'_\ell \sigma'' i'$ is feasible. The jobs ρ'' in the optimal sequence are released on or after time $s(i'')$, so none of the jobs in π'_ℓ nor i' can occur in ρ''. Since $s(i') = s(i'')$ and $p(i') \leq p(i'')$ all jobs in ρ'' can be scheduled after i' in $\pi'_\ell \sigma'' i'$. □

Definition (Type II dominance). Let

$$\pi'_\ell = (i'_1, \ldots, i'_{\ell-1}, j; \text{buffer} = \Phi) \text{ and } \pi''_\ell = (i''_1, \ldots, i''_{l-1}, j; \text{buffer} = \Phi)$$

be elements of $\Pi_\ell^{\mathrm{II}}(j)$. Then π'_ℓ *dominates* π''_ℓ if $s_{\pi'_\ell}(j) \leq s_{\pi''_\ell}(j)$.

Let us denote by $\pi_\ell^{\mathrm{II}}(j)$ a sequence dominating all others in $\Pi_\ell^{\mathrm{II}}(j)$.

Proposition 3. *Let $\pi''_\ell \in \Pi_\ell^{\mathrm{II}}(j)$ correspond to an initial segment of an optimal sequence and let $\pi'_\ell \in \Pi_\ell^{\mathrm{II}}(j)$ dominate π''_ℓ. Then π'_ℓ corresponds to an initial segment of an optimal solution as well.*

Proof. All jobs augmenting π''_ℓ in the optimal sequence can be processed after j for π'_ℓ, since all of them are released after or at time $s_{\pi''_\ell}(j) \geq s_{\pi'_\ell}(j)$ and, consequently, none of them can occur in π'_ℓ. □

From the above three propositions the following result is evident.

Proposition 4. *If there exists an optimal sequence with job j in position ℓ, then there exists an optimal sequence with either $\pi_\ell^{\mathrm{I}}(j)$ or $\pi_\ell^{\mathrm{II}}(j)$ as initial sequence.*

Thus, to construct an optimal sequence, we may proceed as follows: Start out with partial sequences of size 1. We compute a dominating sequence in each set $\Pi_1^{\mathrm{I}}(j)$, $j = 2, 3, \ldots, n$, and $\Pi_1^{\mathrm{II}}(j)$, $j = 1, \ldots, n$.

Note that $\Pi_1^{II}(j)$ contains just one element, which is thus the dominating element $\pi_1^{II}(j) = (j,$ buffer $= \Phi)$.

Suppose that for some $\ell \geq 1$, we have computed a dominating $\pi_\ell^I(j)$ in each nonempty $\Pi_\ell^I(j)$ and a dominating $\pi_\ell^{II}(j)$ in each nonempty $\Pi_\ell^{II}(j)$, $j = 1, \ldots, n$.

Now consider all partial sequences of size $\ell + 1$ that contain any of the $\pi_\ell^I(j)$ or $\pi_\ell^{II}(j)$, $j = 1, \ldots, n$, as 'initial segments' (in the obvious sense). Let $\Pi_{\ell+1}$ denote the set of all these sequences, each of them being of type I or II. Thus, for example, if

$$\pi_\ell^I(j) = (i_1, \ldots, i_{\ell-1}, j; \text{buffer} = i),$$

then both

$$\pi_{\ell+1} = (i_1, \ldots, i_{\ell-1}, j, i; \text{buffer} = \Phi)$$

and

$$\pi'_{\ell+1} = (i_1, \ldots, i_{\ell-1}, j, i_{\ell+1}; \text{buffer} = i)$$

may occur in $\Pi_{\ell+1}$, provided $r(i_{\ell+1}) \geq s(j) + p(j)$.

Moreover, if

$$\pi_\ell^{II}(j) = (i_1, \ldots, i_{\ell-1}, j; \text{buffer} = \Phi)$$

then any sequence of the form

$$\pi_{\ell+1} = (i_1, \ldots, i_{\ell-1}, j, i_{\ell+1}; \text{buffer} = \Phi)$$

with $r(i_{\ell+1}) \geq s(j)$ will occur in $\Pi_{\ell+1}$.

Furthermore, any sequence

$$\pi'_{\ell+1} = (i_1, \ldots, i_{\ell-1}, j, i_{\ell+1}; \text{buffer} = i)$$

will occur in $\Pi_{\ell+1}$, provided $s(j) \leq r(i) < s(j) + p(j) \leq r(i_{\ell+1})$. Note that there are at most $2n$ sequences $\pi_\ell^I(j)$ $(j = 1, 2, \ldots, n)$ and $\pi_\ell^{II}(j)$ $(j = 1, 2, \ldots, n)$. Hence $\Pi_{\ell+1}$ contains at most $O(n^2)$ elements. We partition $\Pi_{\ell+1}$ into disjoint sets

$$\Pi_{\ell+1} \cap \Pi_{\ell+1}^I(j), \quad j = 1, 2, \ldots, n, \quad \text{and} \quad \Pi_{\ell+1} \cap \Pi_{\ell+1}^{II}(j), \quad j = 1, 2, \ldots, n.$$

In each of these sets we compute a dominating sequence and proceed. This is justified by Propositions 2 and 3.

Observe that the dominance relation between two sequences of size ℓ can be decided in time $O(1)$, if we keep track of the starting times of the last jobs. Hence, to compute dominating sequences in each of the above sets takes time $O(n^2)$ in total. The whole algorithm therefore runs in time $O(n^3)$.

We illustrate the algorithm with a small example.

Example. Consider the following 3-job problem: $r(1) = 0$, $r(2) = 1$, $r(3) = 4$, $p(1) = 5$, $p(2) = 2$, $p(3) = 1$. Clearly,

$\Pi_1^I(1) = \emptyset$;

$\Pi_1^I(2) = \{(2; 1)\}$, thus $\pi_1^I(2) = (2; 1)$;

$\Pi_1^I(3) = \{(3; 1), (3; 2)\}$, thus $\pi_1^I(3) = (3; 2)$, since $p(1) > p(2)$, using type I-dominance.

Moreover, we have $\Pi_1^{II}(j) = \{(j; \emptyset)\}$ with $s(j) = r(j)$, $j = 1, 2, 3$, so $\pi_1^{II}(j) = (j; \emptyset)$.

Now for each of these dominating sequences of size 1 we determine their 'successors' of size 2:

$(1; \emptyset) \rightarrow (1, 2; \emptyset)$ and $(1, 3; \emptyset)$, with $s(2) = 5$ and $s(3) = 5$;

$(2; \emptyset) \rightarrow (2, 3; \emptyset)$ with $s(3) = 4$;

$(3; \emptyset) \rightarrow$ no successor;

$(2; 1) \rightarrow (2, 1; \emptyset)$ and $(2, 3; 1)$, with $s(1) = 3$ and $s(3) = 4$;

$(3; 1) \rightarrow (3, 1; \emptyset)$, with $s(1) = 5$.

Collecting the relevant sequences we obtain

$\Pi_2^{\mathrm{I}}(j) = \emptyset$, $j = 1, 2$; $\Pi_2^{\mathrm{I}}(3) = \{(2, 3; 1)\}$;
$\Pi_2^{\mathrm{II}}(1) = \{(2, 1; \emptyset), (3, 1; \emptyset)\}$;
$\Pi_2^{\mathrm{II}}(2) = \{(1, 2; \phi)\}$;
$\Pi_2^{\mathrm{II}}(3) = \{(1, 3; \emptyset), (2, 3; \emptyset)\}$.

Using type II-dominance we obtain

$\pi_2^{\mathrm{I}}(3) = (2, 3; 1)$, $\pi_2^{\mathrm{II}}(1) = (2, 1; \emptyset)$, $\pi_2^{\mathrm{II}}(2) = (1, 2; \emptyset)$, and $\pi_2^{\mathrm{II}}(3) = (2, 3; \emptyset)$.
Of these sequences of size 2 only $\pi_2^{\mathrm{I}}(3)$ has a successor of size 3, i.e. $(2, 3, 1; \emptyset)$, which is the optimal schedule. □

Remark 1. The algorithm is based on dynamic programming. The solution approach is similar to what Denardo [2] calls the reaching method.

Remark 2. Besides the dominance relations introduced, there are several other dominance relations which could be used. However, these do not seem to reduce the complexity of the algorithm. We mention three of them:

a) $\pi'_\ell = (i'_1, \ldots, i'_{\ell-1}, j;\ \text{buffer} = i)$ dominates
$\pi''_\ell = (i''_1, \ldots, i''_{\ell-1}, k;\ \text{buffer} = i)$ if $r(j) + p(j) \le r(k) + p(k)$.

b) $\pi'_\ell = (i'_1, \ldots, i'_{\ell-1}, j;\ \text{buffer} = \emptyset)$ dominates
$\pi''_\ell = (i''_1, \ldots, i''_{\ell-1}, k;\ \text{buffer} = \emptyset)$ if $r(k) \ge s_{\pi'_\ell}(j) + p(j)$.

c) $\pi'_\ell = (i'_1, \ldots, i'_{\ell-1}, j;\ \text{buffer} = i)$ dominates
$\pi''_\ell = (i''_1, \ldots, i''_{\ell-1}, k;\ \text{buffer} = \emptyset)$ if $r(k) \ge r(j) + p(j) + p(i)$.

Remark 3. It should be clear that a similar approach also works for an arbitrary fixed buffer capacity $b \in \mathbb{Z}_+$. To see this, define $\Pi_\ell(j;\ \text{buffer} = i_1, \ldots, i_r)$, $r \le b$, to be the set of partial sequences of size ℓ with job j being the last one processed and jobs $i_1, \ldots, i_r$ residing in the buffer at time $s(j) - \varepsilon$. Within each such class, define a dominance relation based on $s(j)$. In particular let π'_ℓ and π''_ℓ be two partial sequences in the same class, such that the starting times of job j are $s'_\ell(j)$ and $s''_\ell(j)$, respectively. Then π'_ℓ dominates π''_ℓ if $s'_\ell(j) \le s''_\ell(j)$, as in Proposition 3. This follows from the fact that jobs following job j in the sequence with initial segment π''_ℓ are the jobs $\{i_1, i_2, \ldots, i_r\}$ and jobs arriving after $s''(j) \ge s'(j)$. The latter jobs cannot be contained in π'_ℓ and are thus schedulable after j in the sequence with initial segment π'_ℓ. The dominance property implies that for every size ℓ, one has to consider at most $n \cdot n^b = n^{b+1}$ dominant partial sequences of size ℓ. In fact, each sequence $\pi_\ell = (j;\ \text{buffer} = i_1, \ldots, i_r)$ corresponds to a partial schedule where job j is the last (ℓ-th) job processed, with known starting time $s(j)$, and jobs $i_1, \ldots, i_r$ are in the buffer at time $s(j) - \varepsilon$. Each such partial schedule may be extended to a partial schedule of length $\ell + 1$ in essentially two ways, namely, to

(i) a partial schedule $\pi_{\ell+1} = (j';\ \text{buffer} = k_1, \ldots, k_s)$, where $r(j') \ge s(j) + p(j)$. In this case, $s(j') = r(j')$. Furthermore, the buffer jobs $k_1, \ldots, k_r$ are the same as before, i.e. $i_1, \ldots, i_r$ plus possibly some other jobs k_σ with release times $r(k_\sigma) \in [s(j), s(j') - \varepsilon]$.

(ii) a partial schedule $\pi_{\ell+1} = (j';\ \text{buffer} = k_1, \ldots, k_s)$, where $r(j') < s(j) + p(j)$. In this case $s(j') = s(j) + p(j)$. Moreover, j' is either one of the original buffer jobs $i_1, \ldots, i_r$ or a job that arrived during the processing of job j. (In the latter case j' must have been put into the buffer temporarily, implying that $s < b$.) The buffer jobs $k_1, \ldots, k_s$ in this case consist of the old jobs $i_1, \ldots, i_r$ (except possibly job j') plus possibly some jobs k_σ that arrived during the processing of job j.

In any case, it is obvious that each sequence $\pi_\ell = (j;\ \text{buffer} = i_1, \ldots, i_r)$ has at most n^{b+1} possible successors $\pi_{\ell+1} = (j';\ \text{buffer} = k_1, \ldots, k_s)$. Thus, given $O(n^{b+1})$ dominating sequences of size ℓ, we get

$O(n^{2b+2})$ possible successors of length $\ell+1$. We group these into classes $\Pi_{\ell+1}(j'; \text{buffer} = k_1, \ldots, k_s)$ and compute a dominating sequence in each of the $O(n^{b+1})$ classes. This takes time $O(n^{2b+2})$. Summing up for $\ell = 1, \ldots, n$, we get an overall time bound of $O(n^{2b+3})$ for our algorithm. For $b = 1$, this yields $O(n^5)$ compared to the $O(n^3)$ bound we obtained above. The reason is that for $b > 1$, the dominance relation we use is weaker than the ones we used for $b = 1$. It is not clear to us whether there exist in general dominance relations (of 'type I') by means of which one can cut down the number of dominating sequences π_ℓ to $O(n^b)$ and the number of possible successors of each π_ℓ to $O(n^b)$. This would of course result in an $O(n^{2b+1})$ algorithm for the general case.

3. Complexity results for the weighted case

Let us define the decision version of the 1-machine minimum weight loss scheduling problem (denoted by $n \mid 1 \mid b$) as follows:

Instance: An integer $n \in \mathbb{N}$, a set of triples (i.e. jobs) $J = \{(r(j), p(j), w(j)), 1 \le j \le n\}$, with $r(1) \le r(2) \le \cdots \le r(n)$, where $1 \le j \le n$, $(r(j), p(j), w(j)) \in \mathbb{Z}_+^3$, and a number $W \in \mathbb{Z}_+$.

Question: Does there exist a schedule, i.e. a sequence $(j_1, j_2, \ldots, j_k)$ with $\{j_1, j_2, \ldots, j_k\} \subseteq \{1, 2, \ldots, n\}$, such that $\sum_{i=1}^{k} w(j_i) \ge W$, satisfying feasibility conditions

(i) $s(j_i) = \max\{r(j_i), s(j_{i-1}) + p(j_{i-1})\}$, $i = 2, 3, \ldots, k$; $s(j_1) = r(j_1)$;

(ii) the maximum number of pairwise overlapping waiting intervals $[r(j_i), s(j_i)]$, $i = 1, 2, \ldots, k$, is less than or equal to b.

The NP-completeness proof of Theorem 1 below will be based on a reduction from PARTITION (cf. Garey and Johnson [3, p. 60]). Given a set $(a_i \mid i \in I)$ of positive integers with $\alpha := \frac{1}{2}\sum_{i \in I} a_i \in \mathbb{Z}_+$, does there exist a partition $I = T \cup \bar{T}$ such that $\sum_{i \in T} a_i = \alpha$?

Theorem 1. *Problem $n \mid 1 \mid 1$ is NP-complete.*

Proof. It is easy to see that the problem belongs to the class NP. To show that the problem is NP-complete we set $b = 1$ and give a polynomial transformation from PARTITION. Let an instance of partition be given: a finite set $A = \{a_i \in \mathbb{Z}_+ \setminus \{0\} \mid i \in I\}$, an index set $I = \{1, 2, \ldots, n\}$ with $n \in \mathbb{N}$. Let $\alpha = \sum_{i \in I} \frac{1}{2} a_i$ such that $\alpha \in \mathbb{Z}_+$. We construct an instance of $n \mid 1 \mid 1$ as follows. The job set J consists of three subsets J_1, J_2 and J_3 such that

$$J_1 = \{((i-1)\alpha, \alpha, \alpha), i \in I\},$$

$$J_2 = \{((i-1)\alpha, \alpha + a_i, \alpha + a_i), i \in I\},$$

$$J_3 = \{((n+1)\alpha, (n+1)\alpha, (n+1)\alpha), ((n+1)\alpha, (n+1)\alpha, (n+1)\alpha)\}.$$

Finally let $W = 3(n+1)\alpha$. Notice that the weight of a job equals its processing time. At arrival epoch $(i-1)\alpha$, two jobs arrive one with processing time α and the other with processing time $\alpha + a_i$. At time $(n+1)\alpha$ two jobs arrive both having weight $(n+1)\alpha$, which act as 'enforcers'.

Question: Does there exist a feasible schedule attaining maximum weight $W = 3(n+1)\alpha$, or, stated equivalently, does there exist a partition of $J_1 \cup J_2$ allowing a feasible schedule without idle time? The latter follows from the observation that in order to achieve W both enforcer jobs in J_3 must be processed. Thus at time $(n+1)\alpha$ none of the J_1 or J_2 jobs can be in the buffer, which in turn enforce the partition of $J_1 \cup J_2$.

Suppose our partition instance has a solution, i.e. there exists an index set $T \subset I$ such that $\sum_{i \in T} a_i = \sum_{i \in I - T} a_i = \alpha$. Then a solution to $n \mid 1 \mid 1$ can be obtained in the following way: schedule the jobs $\{i \in J_1 \mid i \in I - T\} \cup \{i \in J_2 \mid i \in T\} \cup J_3$ in order of arrival. The feasibility of this schedule follows from the fact that at time $(i-1)\alpha - \varepsilon$ the residual processing time equals $\sum_{j \in K(i)} a_j \le \alpha$, where $K(i) = \{j \in T \mid j < i\}$ for $i = 2, \ldots, n+1$.

Conversely, if $n|1|1$ has a solution, we claim that the partitioning of J_2 among processed and lost jobs induces a solution to PARTITION. First observe that in a feasible schedule the buffer must be empty at time $(n+1)\alpha$. Moreover, there are no idle periods in the time interval $[0, (n+1)\alpha]$. Let the jobs $\{i \in J_2 \mid i \in T\}$, $T \subset I$, be processed. Then there must exist a number k of jobs from J_1 such that

$$k\alpha + \sum_{i \in T} (\alpha + a_i) = (k + |T|)\alpha + \sum_{i \in T} a_i = (n+1)\alpha.$$

Clearly, $k + |T| \leq n-2$ is impossible, since this would imply $\Sigma_{i \in T} a_i \geq 3\alpha$. If $k + |T| = n+1$ we have $\Sigma_{i \in T} a_i = 0$, so $|T| = 0$, which implies that $k = n+1$, a contradiction. There remain two cases to be considered, either $k + |T| = n$ or $k + |T| = n-1$. In the first case we are ready, since this would imply $\Sigma_{i \in T} a_i = \alpha$ and T solves PARTITION. If $k + |T| = n-1$ then $\Sigma_{i \in T} a_i = 2\alpha = \Sigma_{i \in I} a_i$ and hence $|T| = n$, a contradiction. □

Next we will show that the unweighted loss problem for systems with more than one machine and finite capacity buffers is also NP-hard. Consider a system S of two parallel identical machines M_1 and M_2, in which either each machine has its own buffer with capacity one, or in which both machines share a common buffer with capacity two.

Theorem 2. *Given a job set* $J = \{(r(i), p(i), 1), 1 \leq i \leq n\}$, $r(i) \in \mathbb{Z}_+$, $p(i) \in \mathbb{Z}_+$, $n \in \mathbb{N}$, *the problem to determine whether there exists a non-loss schedule for system S is NP-complete.*

Proof. Clearly the problem is in NP. In the present proof, we give a polynomial transformation from PARTITION.

Given $A = \{a_i \in \mathbb{Z}_+ \mid i \in I\}$, $I = \{1, 2, \ldots, n\}$ and $n \in \mathbb{N}$, consider the following instance of our problem, comprised of three subsets of jobs:

$$J = J_1 \cup J_2 \cup J_3, \quad |J_1| = |J_2| = n, \quad |J_3| = 4,$$
$$J_1 = \{((i-1)K, K, 1), i \in I\},$$
$$J_2 = \{((i-1)K, K + a_i, 1), i \in I\},$$
$$J_3 = \{(nK + \alpha, 1, 1)_i, i = 1, 2, 3, 4\},$$

where $\alpha = \frac{1}{2}\Sigma_{i=1}^{n} a_i \in \mathbb{Z}_+$ and $K = 1 + 2\alpha$. Observe that K is chosen such that for the problem containing only the job set $J_1 \cup J_2$ a non-loss schedule exists. If PARTITION has a solution, i.e. $\Sigma_{i \in I} a_i = \Sigma_{i \in I-T} a_i$ for some subset $T \subset I$, then a non-loss schedule for J exists. To see why, assign the jobs

$$\{i \in J_2 \mid i \in T\} \cup \{i \in J_1 \mid i \in I - T\} \cup \{i \in J_3 \mid i = 1, 2\}$$

to M_1 and the jobs

$$\{i \in J_2 \mid i \in I - T\} \cup \{i \in J_1 \mid i \in T\} \cup \{i \in J_3 \mid i = 3, 4\}$$

to M_2. For each machine, process the jobs in nondecreasing order of their release times $r(i)$.

Conversely, suppose our instance of the scheduling problem admits a non-loss solution. Since all J_3 jobs are processed, all J_1 and J_2 jobs are finished at time $nK + \alpha$. All J_1 and J_2 jobs together offer $2nK + 2\alpha$ time units of work to the system. So both machines are busy during the time interval $[0, nK + \alpha]$. Hence, there must exist a subset $T \subset I$ of jobs from J_2 and a number m of jobs from J_1 such that

$$mK + \sum_{i \in T} (K + a_i) = (m + |T|)K + \sum_{i \in T} a_i = nK + \alpha.$$

Now $m + |T| \geq n + 1$ is impossible since then $\sum_{i \in T} a_i \leq \alpha - K < 0$. Also $m + |T| \leq n - 1$ is impossible since this would imply $\sum_{i \in T} a_i \geq \alpha + K > \sum_{i \in T} a_i$. Hence, we conclude that $m + |T| = n$, which yields $\sum_{i \in T} a_i = \alpha$ and, consequently, PARTITION has a solution. □

Remark 4. The above complexity results are based on a transformation from PARTITION which itself is not a strongly-complete problem. The problem considered in Theorem 1 is not strongly-complete since it can be solved in pseudo-polynomial time using a dynamic programming formulation based on a function $f_t(k, \ell)$, $t \in \mathbb{Z}_+$, defined as follows: $f_t(k, \ell)$ = maximum weight of jobs that can be processed in $[0, t]$ such that k is the last finished job, and, at time t, job ℓ either resides in the buffer, or it just enters or leaves the buffer ($\ell = 0$ denoting an empty buffer). A similar approach is likely to work for the two-machine problem considered in Theorem 2. More generally, we conjecture that every minimum loss problem is polynomially solvable, provided there are fixed upper bounds on the number of machines, the processing times and the buffer capacities.

4. Concluding remarks

In the previous sections we proved the following complexity results:
(P_1) The 1-machine problem with unit weights and fixed buffer capacity is polynomially solvable.
(P_2) The 1-machine problem with arbitrary weights and buffer capacity $b = 1$ is NP-hard.
(P_3) The 2-machine problem with weights and a unit buffer per machine is NP-hard.

It was already known that the m-machine problem with arbitrary weights and $b = 0$ (pure loss system) is polynomially solvable, see [1].

The complexity of problem (P_1) for arbitrary b (see Remark 3 of Section 2) is an open problem. We conjecture that it is NP-hard.

Acknowledgement

We thank an anonymous referee for simplifying a previous version of the Proof of Theorem 1.

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] Denardo, E.V., *Dynamic Programming, Models and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[3] Garey, M.R., and Johnson, D.S., *Computers and Intractability. A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA, 1979.
[4] Nawijn, W.M., "Scheduling a single machine delay-loss system", *European Journal of Operational Research* 56 (1992) 364–369.