March, 1978 ESL-P-808

FLOWSHOP SCHEDULING WITH LIMITED TEMPORARY STORAGE

by

Christos H. Papadimitriou*
Paris C. Kanellakis+

ABSTRACT

We examine the problem of scheduling 2-machine flowshops in order to minimize makespan, using a limited amount of intermediate storage buffers. Although there are efficient algorithms for the extreme cases of zero and infinite buffer capacities, we show that all the intermediate (finite capacity) cases are NP-complete. We prove exact bounds for the relative improvement of execution times when a given buffer capacity is used. We also analyze an efficient heuristic for solving the 1-buffer problem, showing that it has a 3/2 worst-case performance. Furthermore, we show that the "no-wait" (i.e., zero buffer) flowshop scheduling problem with 4 machines is NP-complete. This partly settles a well-known open question, although the 3-machine case is left open here.

*Research supported by NSF Grant MCS77-01192

+Research supported by NSF/RANN grant APR76-12036

1. Introduction

In the last few years we have witnessed a spectacular progress towards understanding deterministic multiprocessor scheduling problems of various types. Many interesting problems can be solved by efficient algorithms ([4], [7], [15]), whereas for others it is now understood that such algorithms may very well not exist ([18], [25], [12]). In contrast, single processor scheduling is an area that was considered long ago under control ([5]). For an overview of results in scheduling we recommend [3]; [19], [8] and [14] also stress certain aspects of the area.

Flowshop scheduling is a problem that is considered somehow intermediate between single- and multi-processor scheduling. In the version concerning us here, we are given n jobs that have to be executed on a number of machines. Each job has to stay on the first machine for a prespecified amount of time, and then on the second for another fixed amount of time, and so on. For the cases that the $(j+1)$st machine is busy executing another job when a job is done with the j-th machine, the system is equipped with first-in, first-out (FIFO) buffers, that cannot be bypassed by a job, and that can hold up to b_j jobs at a time (see Figure 1). We seek to minimize the makespan of the job system, in other words, the time between the starting of the first job in the first machine and the end of the last job in the last machine.

Some information had been available concerning the complexity of such problems. In the two-machine case, for example, if we assume that there is no bound on the capacity of the buffer $(b = \infty)$ we can find the optimum schedule of n jobs in $O(n \log n)$ steps using the algorithm of [16]. Notice that, for $m > 2$, the m-machine, unlimited buffer problem

is known to be NP-complete [9]. Also for two machines, when no buffer space is available ($b = 0$, the "no-wait" case) the problem can be considered as a single-state machine problem in the fashion of [7]. As noted by [8], the case of the 2-machine flowshop problem in which b is given positive, finite integer was not as well understood. In fact, in [6] this practical problem is examined, and solutions based on dynamic programming are proposed and tested.

In Section 2 of the present paper we show that all these problems with $0 < b < \infty$ are NP-complete ([18], [1], [12]), and hence, most probably, not susceptible to efficient algorithms. This is somewhat surprising, considering that efficient algorithms do exist for both limiting cases.

Many hard problems are now known to be NP-complete. These include the traveling salesman problem, the satisfiability problem for propositional calculus, and integer programming. The confidence of researchers that these problems cannot be solved by any efficient (polynomial-time) algorithm is due to the facts that (a) no such problem is solvable by any known efficient algorithm, and (b) if one NP-complete problem is solvable by an efficient algorithm, then all NP-complete problems are. Thus, whenever a new problem is added to this elite class, prospective solvers usually turn to less ambitious goals.

One such possible alternative is that of approximation algorithms ([11], [2]); efficient algorithms, that is, producing a solution which is guaranteed to be at most a fixed fraction away from the optimum. We do approach the 1-buffer flowshop problem in this way. With this goal in mind, we prove in Section 3 that using 1 buffer can save up to 1/3 of the

makespan without buffer, and that 1/3 is the best possible such fraction. Finally, in Section 4 we use this idea to show that a simple heuristic (namely scheduling without buffer, and then taking full advantage of the buffer by "squeezing out" as much idle time as possible) produces solutions that are always within 50% of the optimum. We then show that this bound can also be achieved. However, we present simulation results suggesting that the typical performance of our algorithm is of relative error around 4-5%. Our approach can also be extended to b buffer spaces,although the proof is more complicated.

In Section 5 we present results that extend our understanding of the complexity of flowshop scheduling under buffer constraints in another direction: we show that the m-machine zero-buffer problem is NP-complete for $m \geq 4$. As mentioned earlier, the $m = 2$ case can be solved efficiently by using ideas due to Gilmore and Gomory [7] and such "no-wait" problems in general can be viewed as specially structured Traveling Salesman problems [23], [26]. Furthermore, it was known that the problem is hard when m is allowed to vary as a parameter [19]. For fixed m and particularly $m = 3$ the complexity of the problem was an open question [19], [14]. Although our proof for $m \geq 4$ is already very complicated, it appears that settling the $m = 3$ case requires a departure from our methodology.

Finally, in Section 6 we discuss our results, their implications,

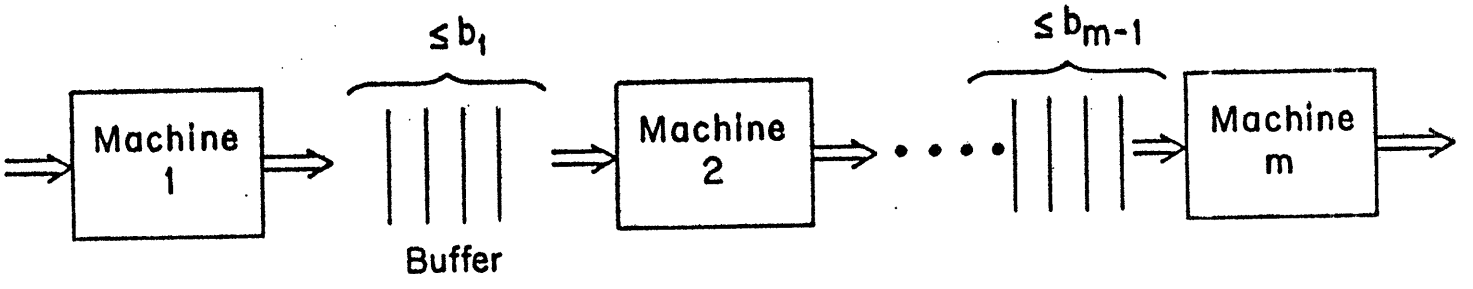


Figure 1

2. The Complexity of Flowshop Scheduling with Buffers

We start by introducing our problem for two machines. Each job is represented by two positive* integers, denoting its execution time requirements on the first and second machine respectively. Now, a feasible schedule with b buffers is an allocation of starting times to all jobs on both machines, such that the following conditions are satisfied:

a) No machine ever executes two jobs at the same time. Naturally, if a job begins on a machine, it continues until it finishes.

b) No job starts on any machine before the previous one ends; no job starts at the second machine unless it is done with the first.

c) No job finishes at the first machine, unless there is buffer space available--in other words there are less than b other jobs that await execution on the second machine.**

d) All jobs execute on both machines in the same order; this restriction comes from the FIFO nature of the buffer.

More formally,

DEFINITION. A job J is a pair (α,β) of positive integers. A feasible schedule with b buffers for a (multi)-set $\mathcal{J} = \{J_1,\ldots,J_n\}$ of jobs (called a job system) is a mapping $S:\{1,\ldots,n\}\times\{1,2\} \to \mathbb{N}$;

---

*For the purpose of clarity in the proofs that follow, we also allow 0 execution times. If a job has 0 execution time for the second machine it is not considered to leave the system after its completion in the first machine. One may disallow 0 execution times, if they seem unnatural by multiplying all execution times by a suitably large integer--say n--and then replacing 0 execution times by 1.

**One may allow the use of the first machine as temporary storage, if no other buffer is available; this does not modify the analysis that follows. In Figure 2 it is demonstrated that this is different from having an extra buffer.

$S(i,j)$ is the starting time of the i-th job on the j-th machine. (The finishing time is defined as $F(i,1) = S(i,1) + \alpha_i$, $F(i,2) = S(i,2) + \beta_i$.) S is subject to the following restrictions

a) $i \neq j \Rightarrow S(i,k) \neq S(j,k)$.

b) Let π_1, π_2 be permutations defined by $i < j \Rightarrow S(\pi_k(i),k) \leq S(\pi_k(j),k)$. Then $\pi_1 = \pi_2 = \pi$ (this is the FIFO rule).

c) $i \neq n \Rightarrow F(\pi(i),k) \leq S(\pi(i+1),k)$.

d) $F(\pi(i),1) \leq S(\pi(i),2)$.

e) $i \leq b + 2 \Rightarrow F(\pi(i-b-1),2) \leq F(\pi(i),1)$.

The makespan of S is $\mu(S) = F(\pi(n),2)$. It should be obvious how the definition above generalizes to m machines.

A feasible schedule is usually represented in terms of a double Ghannt chart as in Figure 2. Here 5 jobs are scheduled on two machines for different values of b, π is the identity permutation. In 2a and 2c a job leaves the first machine when it finishes, whereas in 2b and 2d it might wait. The buffers are used for temporary storage of jobs (e.g., job (3) in 2c spends time τ in the buffer). A schedule without superfluous idle time is fully determined by the pairs (α_i,β_i), π and b; hence finding an optimum schedule amounts to selecting an optimum permutation.

As customary for the purpose of proving NP-completeness we shall first define a corresponding decision problem.

2-machine b-buffer flowshop scheduling ((2,b)-FS)

Given n jobs and integers b and L, is there a feasible schedule S with b buffers such that $\mu(S) \leq L$?

i	a_i	ℓ_i
1	1	2
2	3	3
3	1	1
4	1	1
5	2	1

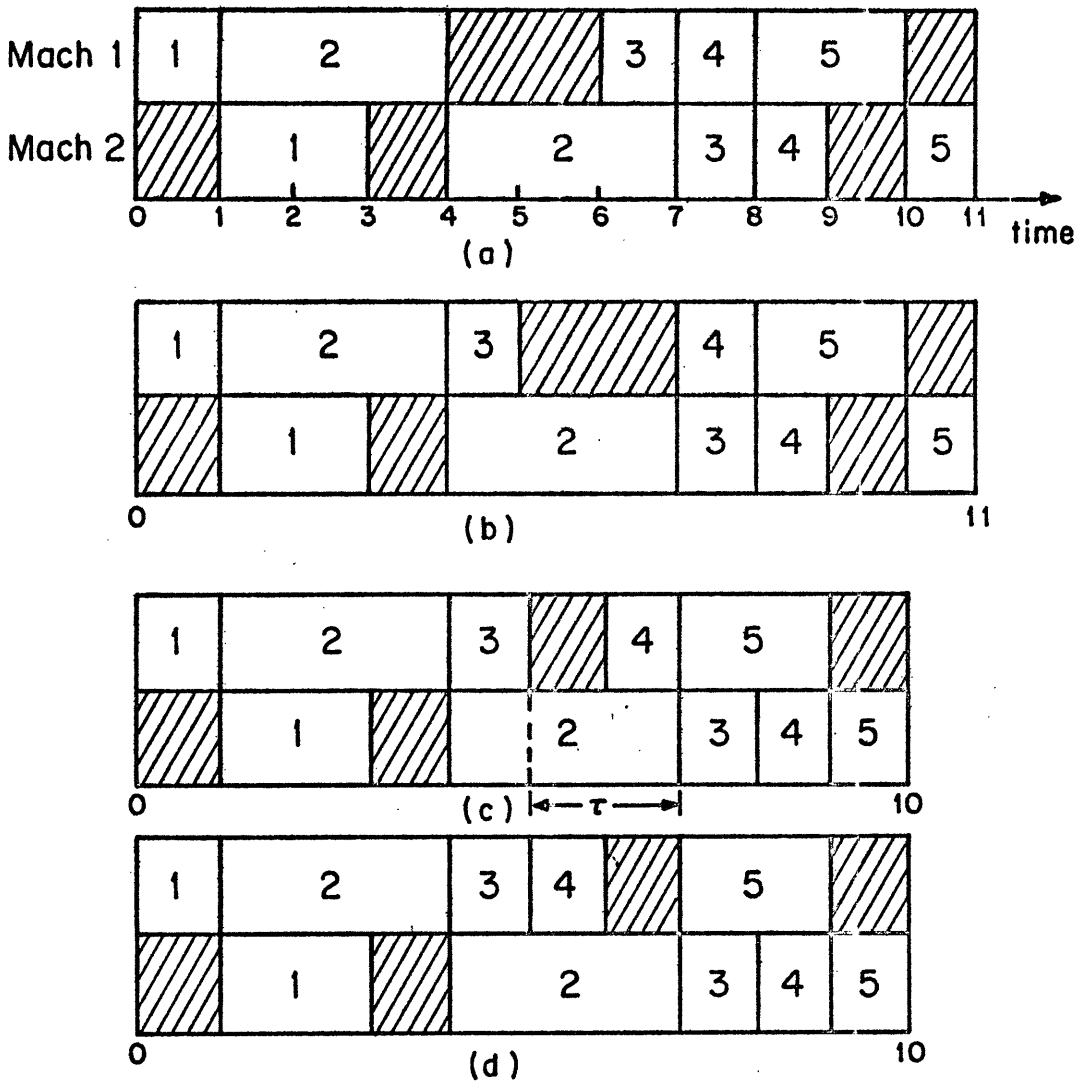


Figure 2

a) b=0

b) b=0

c) b=1

d) b=1

Figure 2

Proving that a problem is NP-complete entails to first showing that it can be solved by a polynomial-time non-deterministic algorithm, and then that a known NP-complete problem is efficiently reducible to it. As usual the first task is routine, since a non-deterministic algorithm could guess

the optimal permutation π, construct the corresponding schedule S, and check that $\mu(S) \leq L$. In our case, the known NP-complete problem that can be reduced to (2,b)-FS is the following.

Three-way matching of integers (3MI)

Given a set A of n integers $A = \{a_1,\ldots,a_n\}$ and a set B of 2n integers $B = \{b_1,\ldots,b_{2n}\}$ is there a partition of B into n pairs $P_i = \{p_{i1},p_{i2}\}$ such that for all i $a_i + p_{i1} + p_{i2} = c$ where $c = 1/n(\Sigma a_i + \Sigma b_j)$ (an integer)?

This problem is known to be NP-complete [12].

THEOREM 1. For all b, $0 < b < \infty$, the (2,b)-FS problem is NP-complete.

Proof. Let us first show that the three-way matching of integers problem reduces in polynomial time to (2,1)-FS. Suppose that we are given an instance $\{a_1,\ldots,a_n\}$, $\{b_1,\ldots,b_{2n}\}$ of the 3MI problem. It is immediately obvious that we can assume that $c/4 < a_i, b_j < c/2$, and that the a_i, b_j's are multiples of 4n; since we can always add to the a_i and b_j's a sufficiently large integer, and then multiply all integers by 4n. Obviously, this transformation will not affect in any way the existence of a solution for the instance of the 3MI problem. Consequently, given any such instance of the 3MI problem, we shall construct an instance I of the (2,1)-FS problem such that I has a schedule with makespan bounded by L iff the instance of 3MI problem were solvable. The instance of the (2,1)-FS problem will have a set $\mathcal{J}$ of 4n+1 jobs, with execution times (α_i, β_i) as follows:

a) We have $n-1$ jobs $K_1,\ldots,K_{n-1}$ with $K_i = (c/2,2)$. Also we have the jobs $K_0 = (0,2)$, and $K_n = (c/2,0)$.

b) For each $1 \leq i \leq 2n$ we have a job $B_i = (1,b_i)$ and for each $1 \leq i \leq n$ we have a job $A_i = (c/2,a_i)$.

L is taken to be $n(c+2)$; this completes the construction of the instance I of the $(2,1)$-FS.

We shall show that I has a schedule S with $\mu(S) \leq L$ iff the original instance of the 3MI problem had a solution. First notice that L equals the sum of all α_i's and also of all β_i's; hence $\mu(S) \leq L$ iff $\mu(S) = L$ and there is no idle time for either machine in S. It follows that K_0 must be scheduled first and K_n last.

We shall need the following lemma:

LEMMA. If for some $j < n$, $S(K_j,2) = k$, then there are integers i_1, $i_2 \leq 2n$ such that $S(B_{i_1},1) = k$, $S(B_{i_2},1) = k+1$.

Proof of Lemma. The lemma says that in any schedule S with no idle times the first two executions on the first machine of jobs $\{B_i\}$ are always as shown in Figure 3a. Obviously, the case shown in Figure 3b--the execution of B_i on the first machine starts and ends in the middle of another job--is impossible, because the buffer constraint is violated in the heavily drawn region. So, assume that we have the situation in 3c. However, since all times are multiples of $4n$ except for the α's of the B_i's and the β's of the K_j's, and since no idle time is allowed in either machine, we conclude that this is impossible. Similarly, the configuration of Figure 3d is also shown impossible. Furthermore, identical arguments hold for subsequent executions of B_i jobs; the lemma follows. □

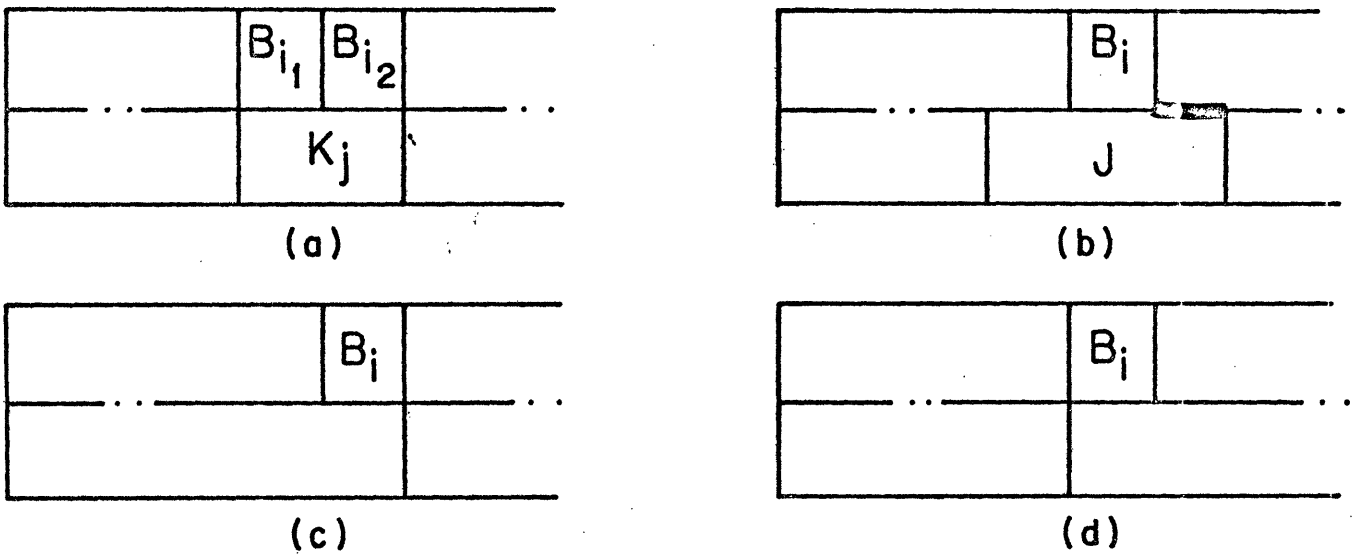


Figure 3

By the lemma, any schedule S of I having no idle times must have a special structure. It has to start with K_0 and then two jobs B_{i_1}, B_{i_2} are chosen. The next job must have an α greater than b_{i_1} but not greater than $b_{i_1} + b_{i_2}$; furthermore it cannot be a K_j job since these jobs must, according to the lemma, exactly precede two B_i jobs and then the buffer constraint would be violated. So we must next execute an A_j job and then a K_k job, because of the inequalities $c/4 < a_i$, $b_i < c/2$. Furthermore, we must finish with the K_k job in the first machine exactly when we finish the A_j job on the second, so that we can schedule two more B jobs (see Figure 4). It follows that any feasible schedule of I will correspond to a decomposition of the set B into n pairs $\{p_{i_1}, p_{i_2}\}$ such that $a_i + p_{i_1} + p_{i_2} = C$.

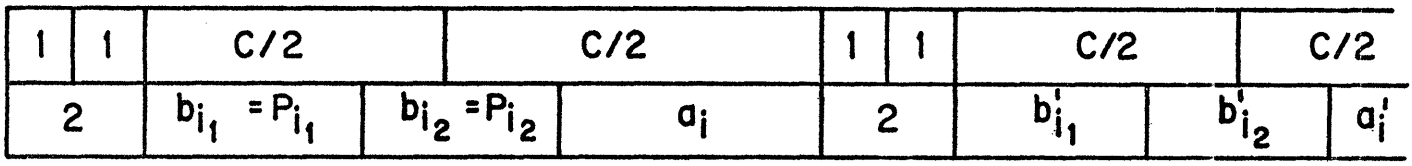


Figure 4

Conversely if such a partition of B is achievable then we can construct a feasible--and without idle times--schedule S by the pattern shown in Figure 4. Hence we have shown that the 3MI problem reduces to $(2,1)$-FS, and hence the $(2,1)$-FS problem is NP-complete.

To complete the proof let us now notice that our argument above generalizes to show that the $(b+2)$MI problem reduces to the $(2,b)$-FS. (In the $(b+2)$MI problem we are given a set A of n integers and a set B of $(b+1)n$ integers; the question is whether B can be partitioned into $(b+1)$ tuples $P_i = (p_{i_1},\ldots,P_{i_{b+1}})$ such that $a_i + \Sigma_{j=1}^{b+1} p_{i_j} = C$. This problem is easily seen to be NP-complete.) Hence we have the Theorem. □

The same technique can be applied to show that minimizing makespan is NP-complete for some other flowshop systems, such as 3-machine flowshops with 0 buffer between machines 1 and 2, and ∞ buffer between machines 2 and 3.

Given a 3MI instance we assume $1 < c/4 < a_i, b_j < c/2 << m$ and we construct a set of jobs J with execution times $(\alpha_i, \beta_i, \gamma_i)$ as follows:

a) We have $n-1$ jobs $K_2,\ldots,K_n$ with $K_i = (m,1,c+1+m)$. Also we have $K_0 = (0,0,1)$, $K_1 = (0,1,c+m+1)$, $K_{n+1} = (m,1,0)$, $K_{n+2} = (1,0,0)$.

b) For each $1 \leq i \leq 2n$ we have a job $B_i = (1,b_i,0)$ and for each $1 \leq i \leq n$ a job $A_i = (0,a_i+m,0)$. L is taken to be $n(c+m+1) + 1$.

It should be noted that $\mu(S) \leq L$ iff there is no idle time on the second and third machines, yet there can be idle time on the first. Decision questions about a job system $\mathcal{J}$ related to whether a number of machines are saturated,(i.e., there is a schedule with no idle time on them) or not will be examined more closely in Section 5.

3. An Upper Bound

Let $\mu_b(\mathcal{J})$ be the shortest possible makespan of a job system $\mathcal{J}$ using b buffers. In this section we show that

$$\sup_{\mathcal{J}} \frac{\mu_0(\mathcal{J})}{\mu_b(\mathcal{J})} = \frac{2b+1}{b+1} \quad .$$

In other words, the use of b buffers can save up to b/2b+1 of the time needed to execute any job system. As in the previous section, we show this first for the $b = 1$ case.

THEOREM 2.

$$\sup_{\mathcal{J}} \frac{\mu_0(\mathcal{J})}{\mu_1(\mathcal{J})} = \frac{3}{2} \quad .$$

Proof. We shall first show that

$$\frac{\mu_0(\mathcal{J})}{\mu_1(\mathcal{J})} \leq \frac{3}{2} \quad .$$

For this purpose, we consider a job system $\mathcal{J}$ and an optimal 1-buffer schedule S of length $\mu_1(\mathcal{J})$. We first notice that we can assume that S is a saturated schedule--that is

$$\mu(S) = \sum_{i=1}^{n} \alpha_i = \sum_{j=1}^{n} \beta_j \quad .$$

To see this one just needs to observe that for any $\mathcal{J}$ and S one can create a job system $\mathcal{J}'$ such that $\mu_1(\mathcal{J}') = \mu_1(\mathcal{J})$, $\mu_0(\mathcal{J}') \geq \mu_0(\mathcal{J})$

and S is a saturated schedule for $\mathcal{J}'$, by "filling in" all idle times of S as shown in Figures 5a, 5b. Given such a saturated schedule S,

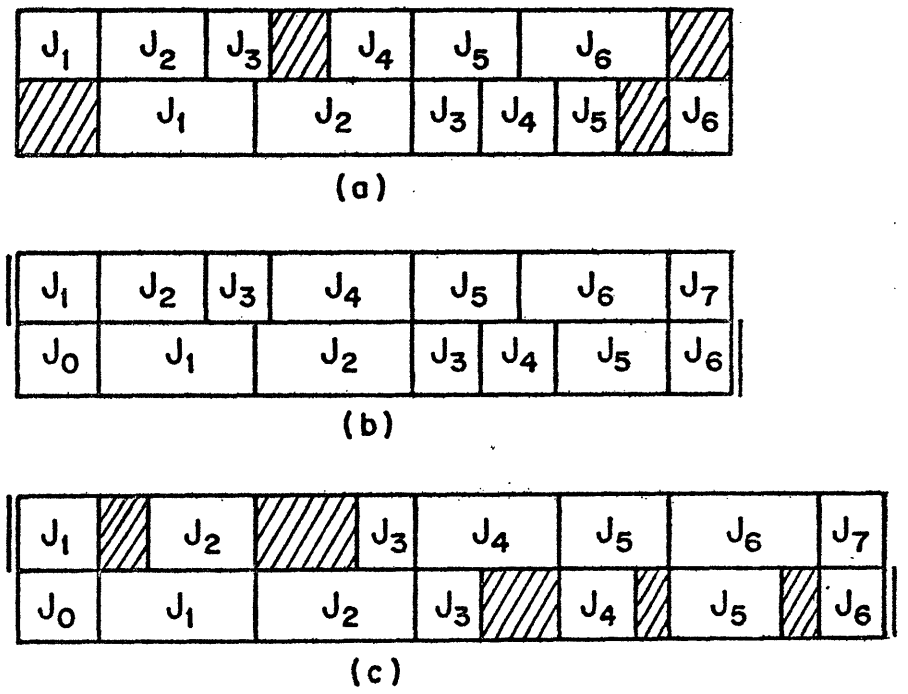


Figure 5

we create a corresponding schedule S' using no buffers and having the same permutation π (Figure 5c). Let us call a maximal set of consecutive--in S'--jobs with no idle time in machine 2 between them a run--in Figure 5c $\{J_0,J_1,J_2,J_3\}$ and $\{J_5\}$ are examples of runs. We shall construct a 0 buffer schedule for $\mathcal{J}'$ with makespan $\leq 3/2\ \mu_1(\mathcal{J}')$; this will then mean that $\mu_0(\mathcal{J}) \leq \mu_0(\mathcal{J}') \leq 3/2\ \mu_1(\mathcal{J}') = 3/2\ \mu_1(\mathcal{J})$. Our construction will examine each run R separately and will consider two cases.

a) The total idle time s in the first machine during run R is less than or equal to $1/2\ \beta(R)$, where $\beta(R)$ is the total execution time on the second machine of jobs in the run R. In this case our construction leaves R intact.

b) $s > 1/2\ \beta(R)$ (see Figure 6a), and hence R consists of $k+1 \geq 2$ jobs. We first note that

$$s = \sum_{i=1}^{k} (\beta_i - \alpha_{i+1}) > \frac{1}{2}\beta(R) ,$$

hence

$$\sum_{i=2}^{k+1} \alpha_i + \beta_{k+1} \leq \frac{1}{2}\beta(R) \quad . \tag{1}$$

We also observe that, in S, the end of α_{k+2} could not have been to the left of the beginning of β_{k+1}, because of the 1-buffer requirement. We conclude that

$$\sum_{i=2}^{k+2} \alpha_i \geq \sum_{j=1}^{k} \beta_k \quad . \tag{2}$$

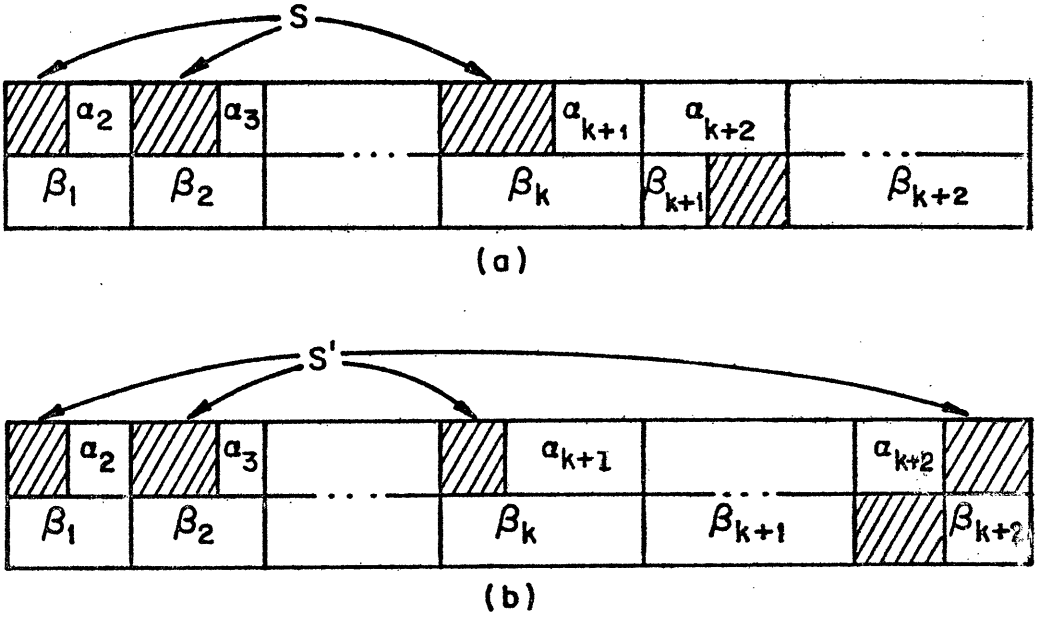


Figure 6

Subtracting (1) from (2) we obtain

$$\alpha_{k+2} \geq \frac{1}{2}\beta(R) \quad . \tag{3}$$

We thus change π--the optimum permutation of $\mathcal{J}'$ for 1 buffer--by putting J_{k+1} in the end. The corresponding 0-buffer schedule is shown in Figure 6b. The total idle time on the first machine due to the jobs in R is now (see Figure 6b).

$$s' = \sum_{j=1}^{k-1} (\beta_j - \alpha_{j+1}) + \beta_k - \min(\beta_k, \alpha_{k+2}) + \beta_{k+1} \leq \beta(R) - \min(\beta_k, \alpha_{k+2}) \ .$$

Two cases:

1. $\beta_k \geq \alpha_{k+2}$. Then $s' \leq \frac{1}{2}\beta(R)$ by (3).

2. $\beta_k \leq \alpha_{k+2}$. In this case we observe that,

$$s' \leq \sum_{i=1}^{k-1} \beta_i + \beta_{k+1} \leq \sum_{i=2}^{k+1} \alpha_i + \beta_{k+1} \leq \frac{1}{2}\beta(R) \quad \text{by (1).}$$

Hence in both cases (a) and (b) our construction succeeds in producing a 0-buffer schedule in which each run R is accountable for machine-1 idle time bounded by $1/2\ \beta(R)$. Hence the total machine-1 idle time is bounded by

$$\frac{1}{2} \sum_{j=1}^{n} \beta_j = \frac{1}{2}\mu_1(\mathcal{J})$$

thus completing the proof of the bound.

It remains to show that this bound is achievable. To do this we consider the job system (for small $\varepsilon > 0$) $\mathcal{J} = \{(\varepsilon,2),(1,\varepsilon),(1,\varepsilon)\}$. The optimal 1-buffer and 0-buffer schedules are shown in Figure 7a and b. The 3/2 ratio is approached as $\varepsilon \to 0$. □

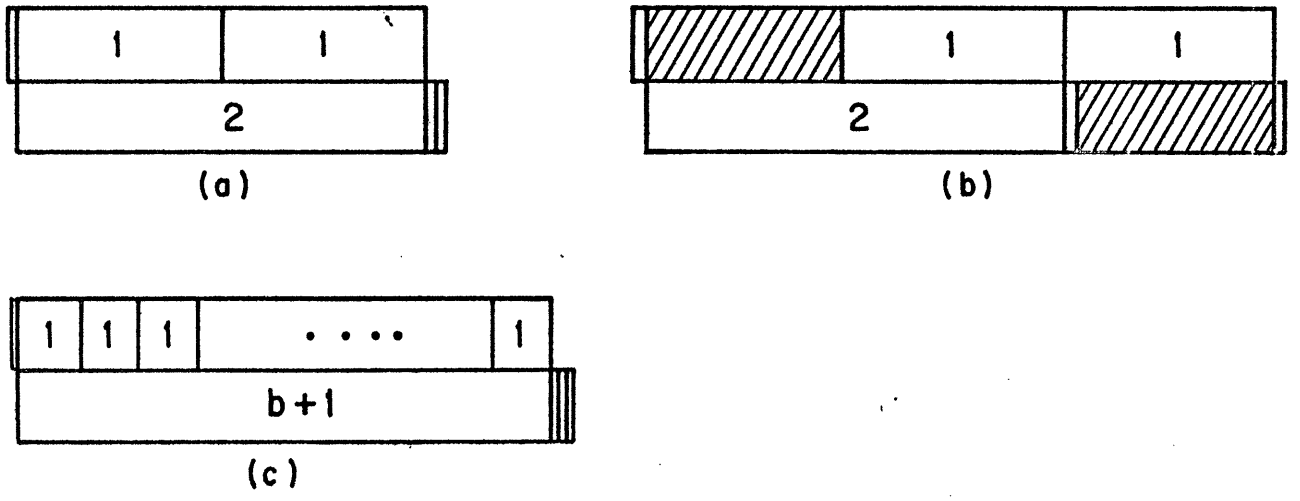


Figure 7

A generalization to any $b > 1$ is possible:

THEOREM 3.

$$\sup_{\mathcal{J}} \frac{\mu_0(\mathcal{J})}{\mu_b(\mathcal{J})} = \frac{2b+1}{b+1} \quad .$$

Proof. Although the argument is similar to the one used for the $b = 1$ case examined above, this time it has to be more complicated. As before we first consider the optimum and--without loss of generality--saturated schedule S, with b buffers, for the job system $\mathcal{J}$. We next

construct the 0-buffer schedule S' corresponding to the same permutation, which, for simplicity, we take to be the identity. We partition the set of jobs into runs, i.e., maximal sets of jobs without intermediate idle time in the second machine in S'. A run with only one job is a singleton; all other runs are proper.

For each run R, let f(R) and $\ell(R)$ be the indices of the first and last jobs of this run, respectively. Also

$$\beta(R) = \sum_{f(R)}^{\ell(R)-1} \beta_j$$

(slightly different from when b=1), $\gamma(R) = \max(0, F(\ell(R),1) - S(f(R),2))$; in other words, $\gamma(R)$ is the total time during which both machines execute jobs in R. (See Figure 8)

For a run R, let C(R) be the set of indices of jobs subsequent to R that execute concurrently with R. Thus $C(R) = \{j > \ell(R): S(j,1) < S(\ell(R),2)\}$; also α_j^R is the portion of J_j that is executed concurrently with R. Thus if $j \notin C(R)$ $\alpha_j^R = 0$; if $j = \max C(R)$ then $\alpha_j^R = S(\ell(R),2) - S(j,1)$ otherwise $\alpha_j^R = \alpha_j$.

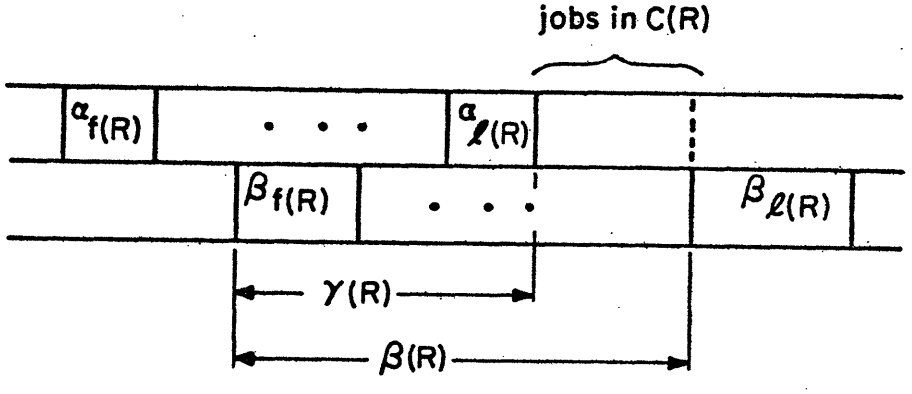


Figure 8

A run as part of schedule S.

LEMMA. For each proper run R there exists an index $i(R) \in R$, and $j(R) \in C(R)$ such that $\beta_{i(R)}, \alpha^{R}_{j(R)} \geq \beta(R)-\gamma(R)/b$, $i(R) \neq \ell(R)$.

Proof of Lemma. During the time from $F(\ell(R),1)$ to $S(\ell(R),2)$--an interval of length $\beta(R)-\gamma(R)$--at most b jobs execute on both machines, because of the buffer constraint. So, at least one of the jobs in R other than the last must satisfy $\beta_{i(R)} \geq \beta(R)-\gamma(R)/b$. Similarly, one of the portions of the jobs in $C(R)$ must satisfy $\alpha^{R}_{j(R)} \geq \beta(R)-\gamma(R)/b$. □

In the sequel we shall assume that $j(R)$, $i(R)$ are chosen such that:

1) $$\beta_{i(R)}, \alpha^{R}_{j(R)} \geq \min\left(\frac{\beta(R)-\gamma(R)}{b}, \frac{\beta(R)+\delta_{j(R)}}{b+1}\right)$$

where $\delta_j = \beta_{j-1}$ if $j = f(R')$ for some run R', and 0 otherwise.

2) $j(R)$ is as small as possible with respect to (1) above.

The existence of such $i(R)$, $j(R)$ is guaranteed by the lemma. Using this lemma, we shall describe a modification of the schedule S'--rather, of the permutation π, currently the identity--and an "accounting scheme" associating to each run R a set of intervals of concurrent execution in the modified schedule of total length at least $\beta(R)/b+1$. We examine all proper runs one-by-one starting from the last. Singletons can be treated in a trivial manner because $\beta(R)-\gamma(R)/b = 0$. Suppose we currently examine R.

Case 1. $j(R) \neq j(R')$ for $R \neq R'$. We may choose to change π from $\pi = \pi_1 i(R) \pi_2 j(R) \pi_3$ to $\pi' = \pi_1 i(R) j(R) \pi_3 \pi_2$; we will then say that R is modified.

Subcase 1.1.

$$\frac{\beta(R) - \gamma(R)}{b} > \frac{\beta(R) + \delta_{j(R)}}{b+1}$$

If $j(R) = f(R')$ we modify R. The accounting scheme assigns in this case to the jobs in $R \cup \{j(R)-1\}$ the interval of concurrent execution resulting from bringing $i(R)$ and $j(R)$ together; it has length at least $\beta(R) + \beta_{j(R)-1}/b+1$. Also notice that $j(R)-1$ is the last job of a run.

If $f(R') \neq j(R) \in R'$ we examine R'. If R' was modified we modify R. The accounting scheme assigns to R the interval run

$$(a^R_{j(R)}, \beta_{i(R)}) \geq \frac{\beta(R)}{b+1} \quad .$$

If R' was not modified we cannot modify R because of the accounting scheme of Subcase 1.2. Thus we do not modify R; the scheme associates with R the execution interval $\alpha^R_{j(R)} \geq \beta(R)/b+1$--which has not been assigned to any run yet.

Subcase 1.2.

$$\frac{\beta(R) + \delta_{j(R)}}{b+1} \geq \frac{\beta(R)-\gamma(R)}{b}$$

hence

$$\gamma(R) + \delta_{j(R)} > \frac{\beta(R) + \delta_{j(R)}}{b+1} \quad .$$

We do not modify R; our accounting scheme assigns to R (and to $j(R)-1$ if $j(R) = f(R')$) the set of concurrent execution intervals of length $\gamma(R) + \delta_{j(R)}$.

Case 2. $j(R-k+1) = \ldots = j(R-1) = j(R)$ for $k > 1$ consecutive runs. In this case we consider all these runs as a unique run $\bar{R}$, and we find a single $i(\bar{R})$ and $j(\bar{R}) = j(R)$. Now suppose that there is a run R', $R-k+2 \leq R' \leq R$, such that $\alpha_{f(R')}^{R-k+1} \geq \beta/b+1$, where

$$\beta = \sum_{j=f(R-k+1)}^{\ell(R)-1} \beta_j \quad .$$

But according to our convention that $j(R)$ is as small as possible, it should be that $j(R-k+1) = f(R') \neq j(R)$ since

$$\frac{\beta}{b+1} \geq \frac{\beta(R-k+1) + \delta_{j(R-k+1)}}{b+1} \quad .$$

Thus, we conclude that

$$\alpha_{f(R')}^{R-k+1} \leq \frac{\beta}{b+1} \qquad \text{for} \quad R' = R-k+2,\ldots,R$$

and hence

$$y = \sum_{R'=R-k+2}^{R} \alpha_{f(R')}^{R-k+1} \leq \frac{(k-1)\beta}{b+1} \quad .$$

Also let

$$x = \sum_{r=R-k+1}^{R-1} \beta_{\ell(R)} \quad ,$$

and

$$\gamma = \max(0, F(\ell(R),1) - S(f(R-k+1),2)).$$

It can be easily seen that, precisely as in the lemma shown above, we can find a job $i(\bar{R})$, where $f(R-k+1) \leq i(\bar{R}) < \ell(R)$, and $i(\bar{R}) \neq \ell(R')$ for all R', such that:

$$\alpha^{\bar{R}}_{j(\bar{R})}, \beta_{i(\bar{R})} \geq \frac{\beta-x-\gamma}{b-k+1} .$$

(We have $\beta-x-\gamma$ time to allocate to $b-k+1$ jobs between $F(\ell(R),1)$ and $S(\ell(R),2)$, where all $\ell(R')$ are in this interval.) Thus we distinguish among two subcases.

Subcase 2.1. $x+\gamma-y < \beta-x-\gamma/b-k+1$; we use the scheme of subcase 1.1.

Subcase 2.2. $x+\gamma-y \geq \beta-x-\gamma/b-k+1$; we do not modify $\bar{R}$.

In either subcase our accounting scheme assigns to $\bar{R}$ concurrent execution of length at least.

$$L = \max\left(x+\gamma-y, \frac{\beta-x-\gamma}{b-k+1}\right) \geq \frac{\beta-y}{b-k+2} .$$

Now, since $y \leq (k-1)\beta/b+1$, as pointed out above, $L \geq \beta/b+1$. Thus, our accounting scheme assigns to the total length of the modified schedule a set of disjoint concurrent execution intervals of total length at least

$$\frac{1}{b+1} \sum_{j \notin F} \beta_j + \sum_{j \in F} \beta_j \geq \frac{1}{b+1} \sum_{j=1}^{|\mathcal{J}|} \beta_j$$

where

$$F = \{j : j = \ell(R) \text{ for some run } R\} - \{j ; j = j(R)-1 \text{ for some run } R\}$$

Thus the modified schedule has total length at most

$$\frac{2b+1}{b+1} \sum_{j=1}^{|\mathcal{J}|} \beta_j ,$$

and therefore

$$\frac{\mu_0(\mathcal{J})}{\mu_b(\mathcal{J})} \leq \frac{2b+1}{b+1} .$$

In order to conclude the proof, we notice that the job system shown in Figure 7c achieves this bound. □

4. An Approximation Algorithm

Consider the following algorithm for obtaining (possibly suboptimal) solutions of the (2,1)-FS problem, for a set of n jobs $\mathcal{J}$.

Algorithm A.

1. Solve the 0-buffer problem for $\mathcal{J}$ using the Gilmore-Gomory algorithm [GG] to obtain a permutation π of $\mathcal{J}$.

2. Schedule $\mathcal{J}$ with 1 buffer using π.

It follows from Theorem 2 that, if $\mu_A(\mathcal{J})$ is the resulting makespan, $\mu_A(\mathcal{J})/\mu_1(\mathcal{J}) \leq 3/2$, since $\mu_A(\mathcal{J}) \leq \mu_0(\mathcal{J})$. However, it does not follow directly that the 3/2 ratio is achievable, because, for the job system $\mathcal{J}$ shown in Figure 7--which was the worst-case job system with respect to Theorem 2--we have $\mu_A(\mathcal{J}) = \mu_1(\mathcal{J})$. The worst-case job system for algorithm A is shown in Figure 9. In Figure 9a we show the optimum 1-buffer schedule with $\mu_1(\mathcal{J}) = 2 + \varepsilon + \delta$. It can be checked that the application of A yields the schedule in Figure 9b, with $\mu_A(\mathcal{J}) = 3 + \delta$. When $\varepsilon < \delta \to 0$ we have an asymptotic ratio of 3/2.

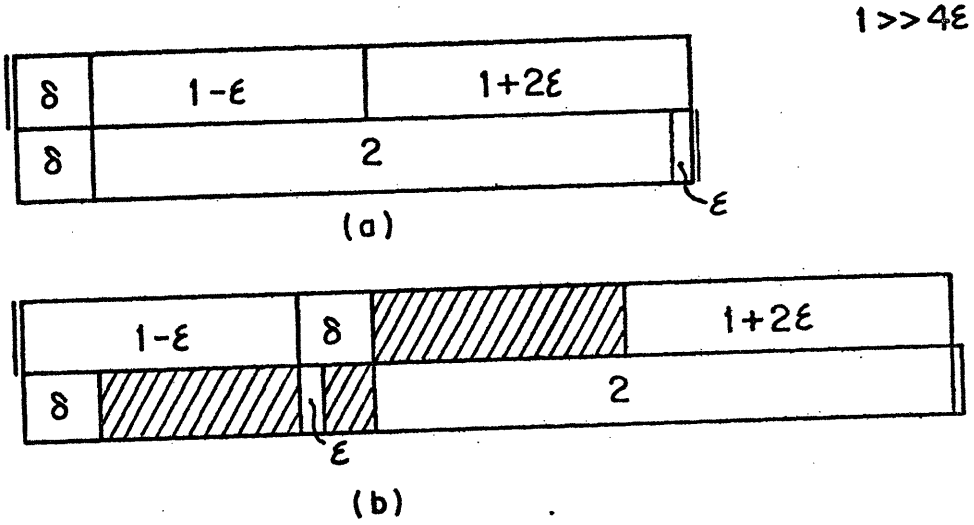


Figure 9

We tested our algorithm on a number of problem instances. For each number of jobs from 4 to 23 we generated 10 job systems among those which have a saturated 1 buffer schedule. The resulting statistics of the relative error are shown in Table 1.

The name heuristic could be used for the (2,b)-FS problem and a similar worst case example, yet the usefulness of the approach decreases as b grows because by basing our schedule on a random permutation we cannot have more than 100% worst case error.

We must remark that the Gilmore Gomory algorithm can be implemented in $0(n \log n)$ as opposed to $0(n^2)$ [7] since the operations in it involve only sorting, calculating n distances and finding a minimum spanning tree in an $0(n)$-edge graph.

TABLE 1

# of jobs	Mean error %	Standard deviation %	worst case error %
4	1.5	5.1	15
5	2.4	8.1	24
6	6.3	9.7	20
7	3.7	5.7	15
8	1.8	2.9	6
9	2.7	4.1	10
10	3.1	4.0	6
11	1.5	3.1	8
12	4.5	5.6	12
13	3.1	4.2	7
14	3.1	4.0	8
15	3.2	3.7	7
16	2.8	3.3	6
17	3.0	4.6	9
18	2.1	3.0	5
19	3.1	4.5	10
20	1.5	2.2	5
21	2.7	3.4	6
22	1.8	2.0	4
23	3.4	3.8	7

5. The Complexity of the "No-Wait" Problem

In certain applications we must schedule flowshops without using any intermediate storage; this is known as the no-wait problem. (For a discussion of this class of problems, see Section 1.) By extending the notation introduced in Section 2 we can define the m-machine no-wait problem as $(m,0)$-FS. In this section we will prove the following

THEOREM 4. The $(4,0)$-FS problem is NP-complete.

For the purposes of this proof, we introduce next certain special kinds of directed graphs. Let $\mathcal{J}$ be an m-machine job system, and let K be a subset of $\{1,2,\ldots,m\}$. The digraph associated with $\mathcal{J}$ with respect to K $D(\mathcal{J};K)$ is a directed graph $(\mathcal{J},A(\mathcal{J};K))$, such that $(J_i,J_j) \in A(\mathcal{J};K)$ iff job J_j can follow job J_i in a schedule S which introduces no idle time in the processors in K (e.g., $k \in K \Rightarrow F(i,k) = S(j,k)$).

The definition of the set of arcs $A(\mathcal{J},K)$ given above could be made more formal by listing an explicit set of inequalities and equalities that must hold among the processing times of the two jobs. To illustrate this point, we notice that if $m=4$ and $K = \{2,3\}$ (Figure 10) the arc (J_1,J_2) is included in $A(\mathcal{J},K)$ iff we have

(1) $\alpha_2 \leq \beta_1$, $\gamma_2 \geq \delta_1$ and $\beta_2 = \gamma_1$.

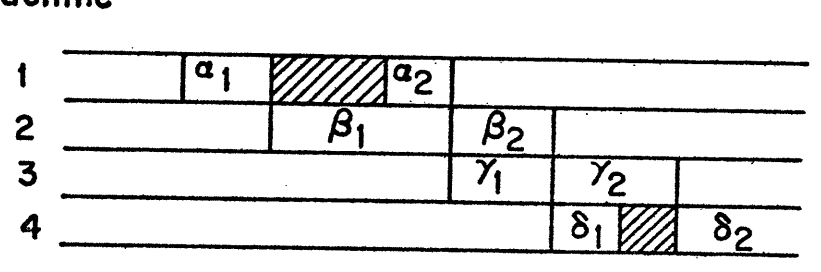


Figure 10

We define $\mathcal{D}(m;K)$ to be the class of digraphs D such that there exists a job system $\mathcal{J}$ with $D = D(\mathcal{J};K)$. We also define the following class of computational problems, for fixed $m > 1$ and $K \in 2^m$:

(m,K)-HAMILTON CIRCUIT PROBLEM

Given an m-machine job system $\mathcal{J}$, does $D(\mathcal{J};K)$ have a Hamilton circuit?

We shall prove Theorem 4 by using the following result:

THEOREM 5. The $(4;\{2,3\})$-Hamilton circuit problem is NP-complete.

We shall prove Theorem 5 by employing a general technique for proving Hamilton path problems to be NP-complete first used by Garey, Johnson and Tarjan [13]. (See also [21], [22].) The intuition behind this technique is that the satisfiability problem is reduced to the different Hamilton path problems by creating subgraphs for clauses on one side of the graph and for variables on the other and relating these subgraphs through "exclusive-or gates" and "or gates" (see Figure 11). We shall introduce the reader to this methodology by the following problem and lemma.

RESTRICTED HAMILTON CIRCUIT PROBLEM

Given a digraph $D = (V,A)$ (with multiple arcs), a set of pairs P of arcs in A and a set of triples T of arcs in A is there a Hamilton circuit C of D such that

a. C traverses <u>exactly one</u> arc from each pair P.

b. C traverses <u>at least one</u> arc from each tuple T.

LEMMA 1. The restricted Hamilton circuit problem is NP-complete.

Proof. We shall reduce 3-satisfiability to it. Given a formula F involving n variables $x_1,\ldots,x_n$ and having m clauses $C_1,\ldots,C_m$ with 3 literals each, we shall construct a digraph D (with possibly multiple arcs), a set of pairs P (two arcs in a pair are denoted as in Figure 12a) and a set of triples T (Figure 12b), such that D has a feasible--with respect to P and T--Hamilton circuit iff the formula is satisfiable.

The construction is a rather straight-forward "compilation." For each variable x_j we have five nodes a_j, b_j, c_j, d_j and e_j, two copies of each of the arcs (a_j,b_j) and (d_j,e_j) and one copy of each of the arcs (b_j,c_j) and (c_j,d_j) (see Figure 11). The "left" copies of (a_j,b_j) and (d_j,e_j) form a pair P. We also connect these sub-digraphs in series via the new nodes f_j. For each clause C_i we have the four nodes u_i, v_i, w_i and z_i and two copies of each of the arcs (u_i,v_i), (v_i,w_i) and (w_i,z_i). Again the "left" copies of these three arcs form a triple in T. These components are again linked in series via some other nodes called y_i (see Figure 11). Also we have the arcs (y_{m+1},f_1) and (f_{m+1},y_1). To take into account the structure of the formula, we connect in a pair P the right copy of (u_i,v_i) with the left copy of (a_j,b_j) if the first literal of C_i is x_j, and to the left copy of (d_j,e_j) if it is $\bar{x}_j$; we repeat this with all clauses and literals. An illustration is shown in Figure 11.

It is not hard to show that D has a feasible Hamilton circuit if and only if F is satisfiable. Any Hamilton circuit C of D must have a special structure: it must traverse the arc (y_{m+1},f_1), and then the

arcs of the components corresponding to variables. Because of the pairs P, if C traverses the left copy of (a_i,b_i), it has to traverse the right copy of (d_i,e_i); we take this to mean that x_i is <u>true</u>. otherwise if the right copy of (a_i,b_i) and the left of (d_i,e_i) are traversed, x_i is <u>false</u>. Then C traverses the arc (f_{m+1},y_1) and the components corresponding to the clauses, one by one. However, the left copies of arcs corresponding to literals are traversed only in the case that the corresponding literal is true; thus, the restrictions due to the triples T are satisfied only if all the clauses are satisfied by the truth assignment mentioned above. (In Figure 11, x_1 = false, x_2 = false, x_3 = true.)

Conversely using any truth assignment that satisfies F, we can construct, as above, a feasible Hamilton circuit for D. This proves the lemma. □

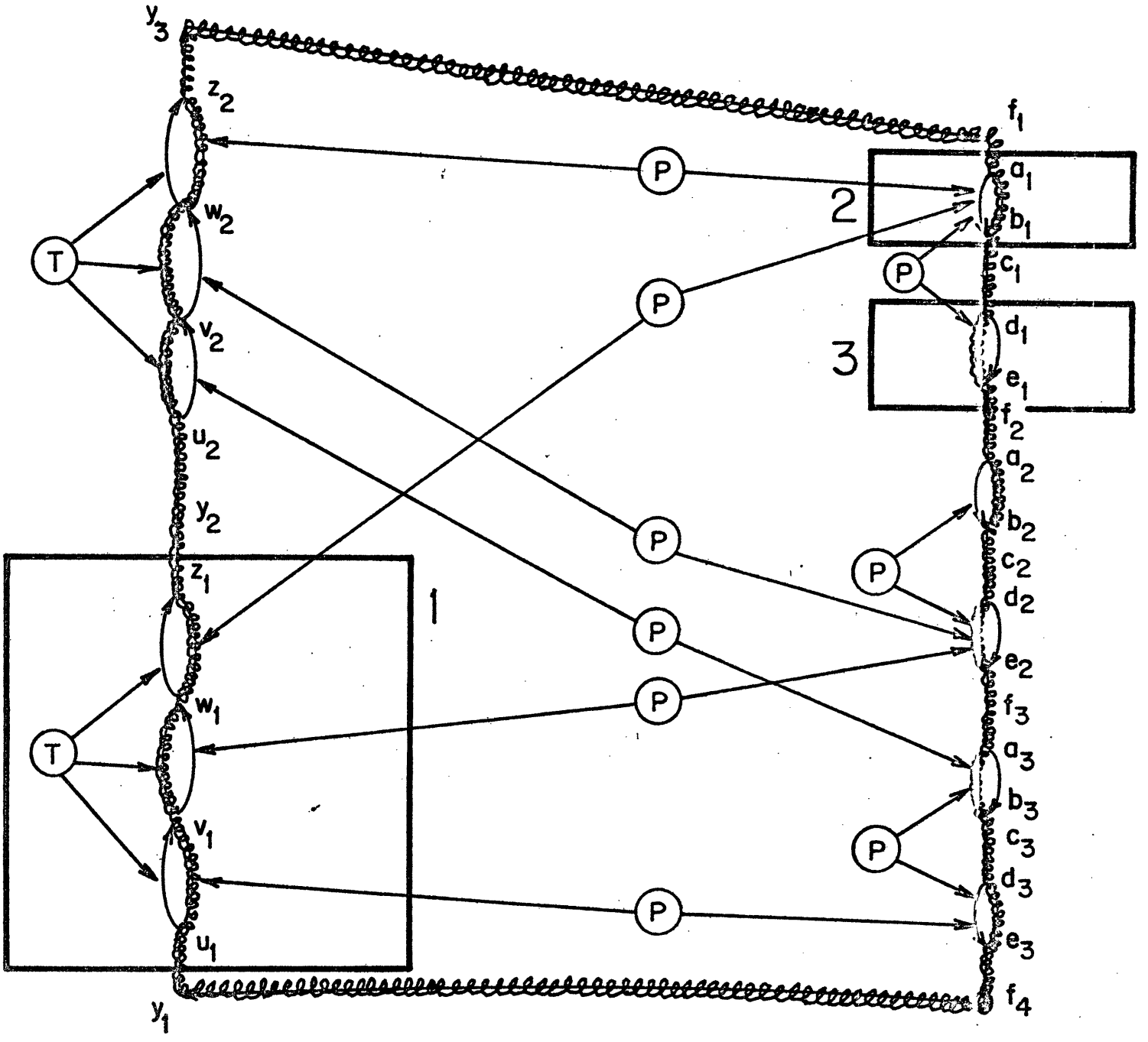
What this lemma (in fact, its proof) essentially says is that for a Hamilton circuit problem to be NP-complete for some class of digraphs, it suffices to show that one can construct special purpose digraphs in this class, which can be used to enforce implicitly the constraints imposed by P (an <u>exclusive-or</u> constraint) and T (an <u>or</u> constraint). For example, in order to show that the unrestricted Hamilton circuit problem is NP-complete, we just have to persuade ourselves that the digraphs shown in Figure 12a and b can be used in the proof of Lemma 1 instead of the P and T connectives, respectively [22]. Garey, Johnson and Tarjan applied this technique to planar, cubic, triconnected graphs [13], and another application appears in [21].

Our proof of Theorem 5 follows the same lines. There are however, several complications due to the restricted nature of the digraphs that concern us here. First, we have to start with a special case of the satisfiability problem.

LEMMA 2. The 3-satisfiability problem remains NP-complete even if each variable is restricted to appear in the formula once or twice unnegated and once negated.

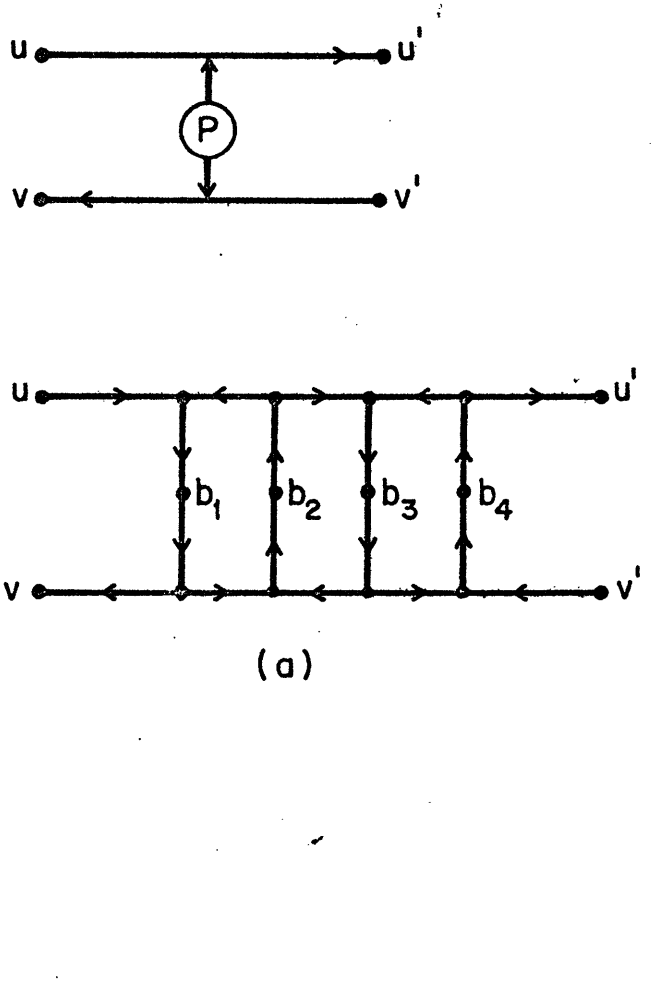
Proof. Given any formula we first modify it so that each variable appears at most three times. Let x be a variable appearing $k > 3$ times in the formula. We replace the first occurrence of x by (the new variable) x_1, the second with x_2, etc. We then add the clauses $(\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \ldots (\bar{x}_k \vee x_1)$--which are, of course $x_1 \equiv x_2 \equiv x_3 \equiv \ldots \equiv x_k$ in conjunctive normal form. We then omit any clause that contains a literal, which appears in the formula either uniformly negated or uniformly unnegated. Finally if x is a variable appearing twice negated, we substitute $\bar{y}$ for x in the formula, where y is a new variable. The resulting formula is the equivalent of the original under the restrictions of Lemma 2. □

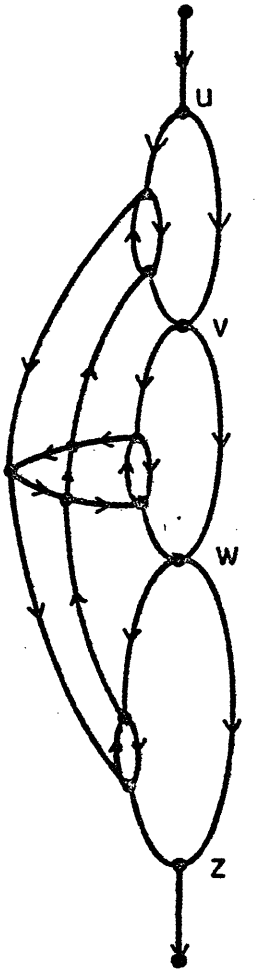
Secondly, realizing special-purpose digraphs in terms of job systems presents us with certain problems. Although our special-purpose digraphs will be similar to those in Figure 12, certain modifications cannot be avoided. A digraph in $\mathcal{D}(4;\{2,3\})$ must be realizable in terms of some job system, so that the inequalities and equations in (1) are satisfied. Care must be taken so that no extra arcs--dangerous to the validity of our argument--are implied in our construction. We shall address this question first.

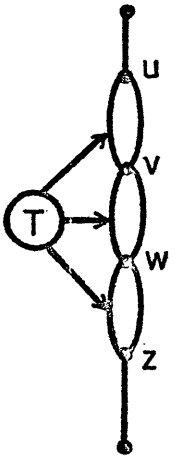


$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3})$$

Figure 11







(b)

Figure 12

Consider a digraph $D = (V,A)$, and a node $b \in V$ such that

a) b has indegree and outdegree one.

b) $(u,b), (b,v) \in A$, where u has outdegree one and v has indegree one.

Then b is called a bond. Removal of all bonds from D divides D into several (weakly connected) components. For example b_1, b_2, b_3, b_4 are bonds in Figure 12a, the y nodes, the f nodes and the c nodes are bonds in Figure 11.

LEMMA 3. If all components of $D = (V,A)$ are in $\mathcal{D}(4;\{2,3\})$, then $D \in \mathcal{D}(4;\{2,3\})$.

Proof. Assuming that each component $F_i (i=1,\ldots,k)$ of D can be realized by a job system $\mathcal{J}_i$, we shall show that D itself can be realized by a job system $\mathcal{J}$. For each $\mathcal{J}_i$ we modify the execution times as follows: we multiply all execution times by $|V| \cdot k$ and then add $(i-1)|V|$ to each; this obviously preserves the structure of each F_i, but has the effect that there are no cross-component arcs, because all components have now different residues of execution times modulo $k \cdot |V|$ and hence the $\beta_i = \gamma_j$ equality cannot hold between nodes from different components.

Next we have to show how all bonds can be realized. Let b_j be a bond of D such that $(u,b_j), (b_j,v) \in A$. Suppose that the jobs realizing u and v have execution times $(\alpha_u,\beta_u,\gamma_u,\delta_u)$ and $(\alpha_v,\beta_v,\gamma_v,\delta_v)$, respectively. Since u has outdegree one and v has indegree one we can arrange it so that β_v and γ_u are unique. Thus b_j can be realized by the job $(0,\gamma_u,b_v,0)$. Repeating this for all

bonds we end up with a realization of D in terms of 4-machine jobs with saturated second and third machine. The Lemma follows. □

We shall now proceed with the construction of the job system $\mathcal{J}$, corresponding to a digraph D, starting from any Boolean formula F, as required for the proof of Theorem 5. As mentioned earlier, the construction is essentially that pictured in Figure 11 and our P- and T-digraphs are similar--although not identical--to the ones shown in Figures 12a and 12b. Lemma 3 enables us to perform the construction for each component separately. The components of D do not exactly correspond to the P- and T-digraphs: They correspond to portions of the digraph in Figure 11 such as the ones shown within the boxes 1, 2 and 3. They are, indeed, components of D, since the c, f, y nodes are bonds as are the b_1, b_2, b_3, b_4 nodes of the P-digraph in Figure 12a.

In Figure 13a we show the component corresponding to each clause of F, as well as its realization by a job system $\mathcal{J}$ shown in Figure 13b. We omit here the straight-forward but tedious verification that, indeed, the component shown is $D(\mathcal{J};\{2,3\})$. We only give the necessary inequalities between the processing times of tasks corresponding to nodes $\{1,2,3,\ldots,10\}$. Each of the quadruples of nodes $(2,3,4,5)$, $(12,13,14,15)$ and $(22,23,24,25)$ is the one side of a P-digraph, and they are to be connected, via appropriate bonds, to the quadruples associated with the literals of the clause.

In Figure 14a we show the component that corresponds to an unnegated variable occurring twice in F. Again the quadruples $(2,3,4,5)$, $(6,8,7,9)$, $(10,11,12,13)$ are parts of P-digraphs. The first two are

to be connected via bonds to the components of the clauses in which this variable occurs. The third quadruple is to be connected by bonds with the component of the negation of the same variable. Notice, that this component is in $\mathcal{D}(4;\{2,3\})$ as demonstrated in Figure 14b.

The lower part of 14a shows the component that corresponds to negations of variables and is realizable in a similar manner as in 14b.

The remaining argument is to the effect that copies of these three components, when properly connected via bonds as shown in Figure 11, function within their specifications. Although certain arcs that we had to add in order to make D realizable by 4-machine jobs (such as the lines (9,6) and (13,4) in Figure 14a) may render it slightly less obvious, the argument of Lemma 1 is valid. First, it is well to observe that lines such as (9,6) in Figure 14a and (5,2) in Figure 13a can never participate in a Hamilton circuit and are therefore irrelevant. Secondly with a little more attention the same can be concluded for arcs like (13,8) and (13,4) of Figure 14a. It is then straight-forward to check that the remaining digraph behaves as desired. In other words, for each Hamilton circuit c and each variable x, either the arc (1,14) (Figure 14a), corresponding to x, or the arc (15,16) (Figure 14a), corresponding to $\bar{x}$, is traversed. The former means that x is <u>false</u>, the latter that it is <u>true</u>, then only clauses having at least one literal true shall have the corresponding nodes 9, 10, 11 (Figure 13a) traversed. Thus, a Hamilton circuit exists in D if and only if F is satisfiable and the sketch of our proof is completed. □

Now we can prove Theorem 4.

Proof of Theorem 4. We shall reduce the $(4;\{2,3\})$-Hamilton circuit problem to it. Let $\mathcal{J}$ be a job system constituting an instance of this problem. It is evident from the proof of Theorem 5 that we can assume that $D(\mathcal{J};\{2,3\})$ has at least one bond, J_b, having execution times unlike any other execution times of jobs in $\mathcal{J}$. Let (J_1,J_b), $(J_b,J_2) \in D(\mathcal{J};\{2,3\})$, where $J_1 = (\alpha_1,\beta_1,\gamma_1,\delta_1)$ and $J_2 = (\alpha_2,\beta_2,\gamma_2,\delta_2)$. We create the job system $\mathcal{J}' = \mathcal{J} - \{J_b\} \cup S$, where

$$S = \{(0,\alpha_2,\beta_2,\gamma_2),\ (0,0,\alpha_2,\beta_2),\ (0,0,0,\alpha_2),\ (\beta_1,\gamma_1,\delta_1,0),$$
$$(\gamma_1,\delta_1,0,0),\ (\delta_1,0,0,0)\}\ .$$

It should be obvious that $D(\mathcal{J},K)$ has a Hamilton circuit if and only if $\mathcal{J}'$ has a no-wait schedule with makespan $\sum_{j\in\mathcal{J}'} \beta_j$ or less. □

Since the m-machine no-wait problem can be reduced to the (m+1)-machine no wait problem we conclude.

COROLLARY. The m-machine no-wait problem is NP-complete for $m \geq 4$. □

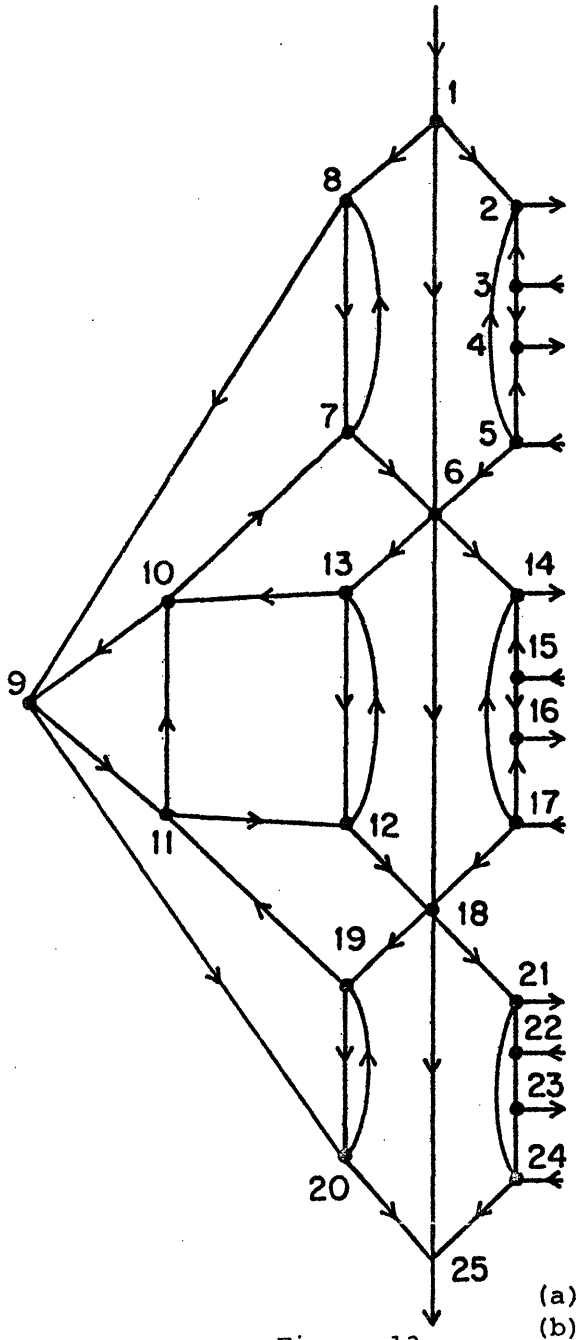


Figure 13

(a) component for clause
(b) table of values
(c) inequalities and lattice of β's and δ's for nodes 1 to 10

(a)

$\beta_1, \beta_3, \beta_5 \geq \alpha_2$

$\delta_1, \delta_3, \delta_5 \leq \gamma_2 \leq \delta_7$

$\beta_3, \beta_5 \geq \alpha_4$

$\delta_3, \delta_5 \leq \gamma_4 < \delta_7, \delta_1$

$\beta_1, \beta_5, \beta_7 \geq \alpha_6 > \beta_3$

$\delta_1, \delta_5, \delta_7 \leq \gamma_6$

$\beta_1, \beta_7 \geq \alpha_8 > \beta_3, \beta_5$

$\delta_1, \delta_7 \leq \gamma_8$

$\beta_{10}, \beta_8 \geq \alpha_7$

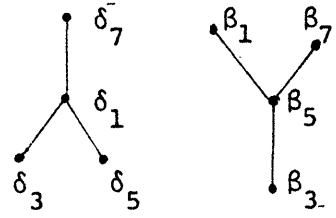
$\delta_{10}, \delta_8 \leq \gamma_7$

$\beta_{10}, \beta_8 \geq \alpha_9$

$\delta_{10}, \delta_8 \leq \gamma_9$

$\gamma_1 = \beta_2 = \gamma_3 = \beta_4 = \gamma_5 = \beta_6 = \gamma_7 = \beta_8$

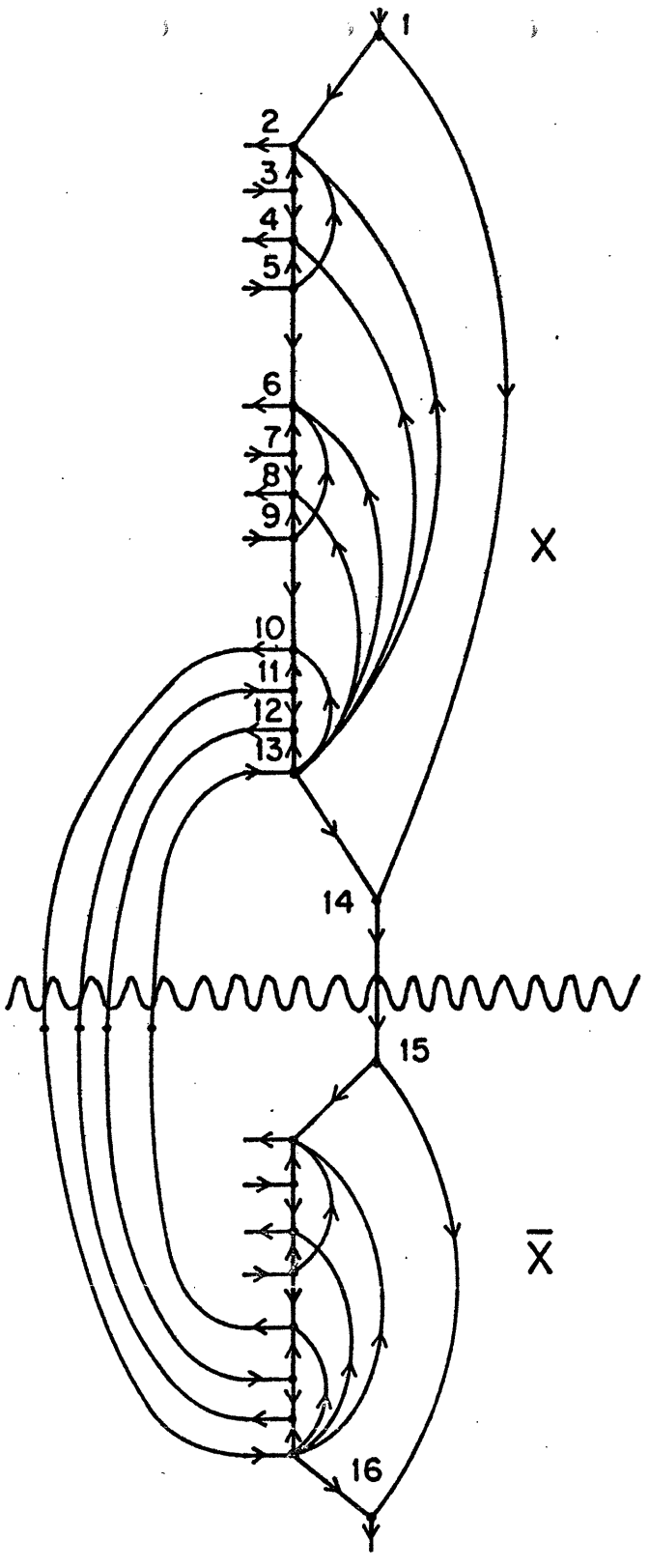
$\gamma_8 = \beta_7 = \beta_9 = \gamma_{10}$



(c)

(b)

	α	β	γ	δ
1	0	15	100	10
2	4	100	14	0
3	0	5	100	5
4	4	100	9	0
5	0	10	100	5
6	9	100	200	10
7	0	16	100	15
8	11	100	16	0
9	0	16	17	0
10	0	18	16	0
11	0	17	18	0
12	0	17	200	15
13	11	200	17	0
14	4	200	14	0
15	0	5	200	5
16	4	200	9	0
17	0	10	200	5
18	9	200	300	10
19	11	18	300	0
20	0	300	18	15
21	4	300	14	0
22	0	5	300	5
23	4	300	9	0
24	0	10	300	5
25	9	300	400	0



$\beta_1,\beta_3,\beta_5,\beta_{13} \geq \alpha_2 > \beta_7,\beta_9,\beta_{11}$

$\delta_1,\delta_3,\delta_5,\delta_{13} \leq \gamma_2$

$\beta_3,\beta_5,\beta_{13} \geq \alpha_4 > \beta_7,\beta_9,\beta_{11}$

$\delta_3,\delta_5,\delta_{13} \leq \gamma_4 < \delta_1$

$\beta_5,\beta_7,\beta_9,\beta_{13} \geq \alpha_6 > \beta_{11}$

$\delta_5,\delta_7,\delta_9,\delta_{13} \leq \gamma_6 < \delta_1,\delta_3$

$\beta_7,\beta_9,\beta_{13} \geq \alpha_8 > \beta_{11}$

$\delta_7,\delta_9,\delta_{13} \leq \gamma_8 < \delta_1,\delta_3,\delta_5$

$\beta_9,\beta_{11},\beta_{13} \geq \alpha_{10}$

$\delta_9,\delta_{11},\delta_{13} \leq \gamma_{10} < \delta_1,\delta_3,\delta_7,\delta_5$

$\beta_{11},\beta_{13} \geq \alpha_{12}$

$\delta_{11},\delta_{13} \leq \gamma_{12} < \delta_3,\delta_5,\delta_7,\delta_9,\delta_1$

$\beta_1,\beta_{13} \geq \alpha_{14} > \beta_3,\beta_5,\beta_7,\beta_9,\beta_{11}$

$\delta_1,\delta_{13} \leq \gamma_{14}$

$\gamma_1=\beta_2=\gamma_3=\beta_4=\gamma_5=\beta_6=\gamma_7=$

$=\beta_8=\gamma_9=\beta_{10}=\gamma_{11}=\beta_{12}=\gamma_{13}=\beta_{14}$

(a)

	α	β	γ	δ
1	0	11	15	13
2	6	15	14	0
3	0	9	15	11
4	6	15	12	0
5	0	7	15	9
6	2	15	10	0
7	0	5	15	7
8	2	15	8	0
9	0	3	15	5
10	0	15	6	0
11	0	1	15	3
12	0	15	4	0
13	0	13	15	1
14	10	15	14	0

(b)

β_{13}	δ_1
β_1	δ_3
β_3	δ_5
β_5	δ_7
β_7	δ_9
β_9	δ_{11}
β_{11}	δ_{13}

(c)

(a) components for variables
(b) table of values
(c) inequalities and lattice of β's and δ's

FIGURE 14

5. Discussion

We saw that the complexity of scheduling two-machine flowshops varies considerably with the size of the available intermediate storage. Two classical results imply that when either no intermediate storage or unlimited intermediate storage is available there are efficient algorithms to perform this task. When we have a buffer of any fixed finite size however, we showed that the problem becomes NP-complete.

We showed that using 1 buffer can save up to 1/3 of the makespan required without buffer and this generalized to b buffers. We have used this fact to develop a heuristic, which has a 50% worst case behavior for the (2,1)-FS problem but appears to perform much better (4-5% error) on typical problem instances. We notice that our simulation results suggest that our algorithm performs better than the heuristic reported in [6] for small b.

The formalism in [11], suggests that the 1-buffer 2 machine flowshop problem is, like the TSP and 3MI, strongly NP-complete; that is, unless $\mathcal{P} = \mathcal{NP}$ there can be no uniform way of producing ε-approximate solutions by algorithms polynomial in n and $1/\varepsilon$. The same implications hold for the problems in Section 5, since, as the reader can check, the size of the execution times used in the construction remains bounded by a polynomial in n, the number of jobs.

Since the results in Section 5 indicate that fixed size no-wait flowshop problems are NP-complete and because these problems are actually Asymmetric Traveling Salesman (ATSP) problems, which have distances obeying the triangle-inequality, they provide a strong motivation for good heuristics for the ATSP. The most successful known heuristic [20]

works for the symmetric case. Notice also that no general approximation algorithm of any fixed ratio is known for the triangle inequality TSP in contrast with the symmetric [2]. In [17] we develop a methodology for asymmetric TSP's paralleling that of [20], so as to cope with the intricate pecularities of the asymmetric case.

Our results of Section 5 leave only one open question, as far as no-wait problems are concerned: the 3-machine case. Admittedly this problem--and the generous prize that comes with its solution [19] --was the original goal of our efforts. We conjecture that this problem is NP-complete, although we cannot see how to prove this without a drastic departure from the methodology used here. One may wish to show that the Hamilton circuit problem is NP-complete for $\mathcal{D}(3;K)$ for some $K \neq \phi$. Now, if $|K| = 2$ the corresponding problem is polynomial. The $|K| = 3$ case and, in general, the Hamilton problems for $\mathcal{D}(m;\{1,2,...m\})$ are equivalent to searching for Euler paths in graphs in which the jobs are represented by arcs and the nodes are the "profiles" of jobs in the Ghannt chart [24]. Consequently, this class of problems can be solved in linear time. This leaves us with the $|K| = 1$ case; the authors have different opinions regarding the tractability of this problem.

We conclude by examining how much our assumption that the buffer is FIFO affects the resulting scheduling problem. Removing this assumption would correspond to removing line (b) from the definition of the (2,b)-FS problem. In Figure 15 we show a job system that fares slightly better when the FIFO assumption is removed. We conjecture that removing the FIFO assumption results, at best, in negligible gains for the $b = 1$ case.

job #	α_i	β_i
1	0	100
2	80	40
3	10	1
4	50	9
5	5	0
6	5	0

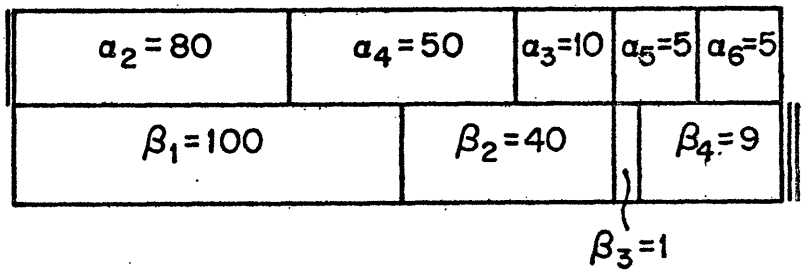


Figure 15

b=1 $\quad \pi_1 \neq \pi_2$

For every $\pi = \pi_1 = \pi_2$ we would be forced to introduce idle time.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., (1974).

[2] N. Christofides, "Worst-Case Analysis of an Algorithm for the TSP," Carnegie-Mellon Conf. on Algorithms and Complexity, Pittsburg, (1976).

[3] E. G. Coffman, Jr. (editor),"Computer and Job-Shop Scheduling Theory," Wiley, New York, (1976).

[4] E. G. Coffman, Jr., R. L. Graham, "Optimal Scheduling for Two-Processor Systems," Acta Informatica, 1, 3 (1972), pp. 200-213.

[5] R. W. Conway, W. L. Maxwell, L. W. Miller, "Theory of Scheduling," Addison Wesley, Reading, Mass. (1967).

[6] S. K. Dutta, A. A. Cunningham, "Sequencing Two-Machine Flowshops with Finite Intermediate Storage," Management Sci., 21 (1975), pp. 989-996.

[7] P. C. Gilmore, R. E. Gomory, "Sequencing a One-State Variable Machine: A Solvable Case of the Traveling Salesman Problem," Operations Res., 12 (1964), pp. 655-679.

[8] M. J. Gonzales, Jr., "Deterministic Processor Scheduling," Computing Surveys, 9, 3 (1977), pp. 173-204.

[9] M. R. Garey, D. S. Johnson, "Complexity Results for Multiprocessor Scheduling Under Resource Constraints," Proceedings, 8th Annual Princeton Conference on Information Sciences and Systems (1974).

[10] M. R. Garey, D. S. Johnson, R. Sethi, "The Complexity of Flowshop and Jobshop Scheduling," Math. of Operations Research, 1, 2(1974), pp. 117-128.

[11] M. R. Garey, D. S. Johnson, "Strong NP-Completeness Results: Motivation, Examples and Implications," to appear (1978).

[12] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, 1978 (to appear).

[13] M. R. Garey, D. S. Johnson, R. E. Tarjan, "The Planar Hamiltonian Circuit Problem is NP-Complete," Siam J. of Computing, 5, 4(1976), pp. 704-714.

[14] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," to appear in Annals of Discrete Math. (1977).

[15] T. C. Hu, "Parallel Sequencing and Assembly Line Problems," Operations Research, 9, 6 (1961), pp. 841-848.

[16] S. M. Johnson, "Optimal Two-and-Three-Stage Production Schedules," Naval Research and Logistics Quarterly 1, 1 (1954), pp. 61-68.

[17] P. C. Kanellakis, C. H. Papadimitriou, "A Heuristic Algorithm for the Asymmetric Traveling Salesman Problem," in preparation.

[18] R. M. Karp, "Reducibility Among Combinatorial Problems," Complexity of Computer Computation, R. E. Miller and J. W. Thatcher (Eds.), Plenum Press, New York, (1972), pp. 85-104.

[19] J. K. Lenstra, "Sequencing by Enumerative Methods," Mathematisch Centrum, Amsterdam (1976).

[20] S. Lin, B. W. Kernighan, "An Effective Heuristic for the Traveling Salesman Problem," Operations Research, 21, 2 (1973), pp. 498-516.

[21] C. H. Papadimitriou, "The Adjacency Relation on the Traveling Salesman Polytope is NP-Complete," Mathematical Programming, 1978 (to appear).

[22] C. H. Papadimitriou, K. Steiglitz, "Combinatorial Optimization Algorithms," book in preparation, 1978.

[23] S. S. Reddi, C. V. Ramamoorthy, "On the Flowshop Sequencing Problem with No Wait in Process," Operational Res. Quart. 23, (1972), pp. 323-331.

[24] R. L. Rivest, private communication.

[25] J. D. Ullman, "NP-Complete Scheduling Problems," J. Comput. System Sci., 10, (1975), pp. 384-393.

[26] D. A. Wismer, "Solution of the Flowshop Scheduling Problem with No Intermediate Queues," Operations Res., 20, (1972), pp. 689-697.