ALGORITHMS FOR A SCHEDULING APPLICATION
OF THE ASYMMETRIC TRAVELING SALESMAN PROBLEM

by

Paris-Christos Kanellakis

S.B., National Tech. Univ. of Athens, Greece, (1976)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1978

Signature of Author..
Department of Electrical Engineering
and Computer Science, May 24, 1978

Certified by..
Thesis Supervisor

....
Thesis Supervisor

Accepted by..
Chairman, Department Committee on Graduate Students

ALGORITHMS FOR A SCHEDULING APPLICATION OF THE
ASYMMETRIC TRAVELING SALESMAN PROBLEM

by

Paris Christos Kanellakis

Submitted to the Department of Electrical Engineering and Computer Science on May 24, 1978, in Partial Fulfillment of the Requirements for the Degree of Master of Science.

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 analyze an efficient approximation algorithm for solving the finite buffer problem. 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. All the above problems are formulated as or approximated by Asymmetric Traveling Salesman Problems (A-TSP). A methodology is developed to treat (A-TSP)'s.

Thesis Supervisor: Ronald L. Rivest
Title: Associate Professor of EECS

Thesis Supervisor: Michael Athans
Title: Professor of EECS, Director ESL

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Prof. Christos H. Papadimitriou of Harvard for his help and collaboration that made this research possible. I am greatly indebted to Prof. Ronald L. Rivest who supervised this work with great understanding and many helpful suggestions. I would also like to express my deep appreciation to Prof. Michael Athans, whose encouragement and constant support were invaluable.

Finally, I would like to thank Ms. Margaret Flaherty for the excellent typing of this manuscript.

This research was carried out at the M.I.T. Electronic Systems Laboratory with partial support extended by the National Science Foundation grant NSF-APR-76-12036.

TABLE OF CONTENTS

	Page
ABSTRACT	2
ACKNOWLEDGEMENTS	3
TABLE OF CONTENTS	4
CHAPTER 1: INTRODUCTION AND PROBLEM DESCRIPTION	6
1.1 Motivation	6
1.2 Problem Description	9
1.2.1 Discussion	9
1.2.2 The Finite Buffer Problem	10
1.2.3 The "No-Wait" Problem	14
1.2.4 The Significance of Constraints	20
1.3 The General Scheduling Model and Extensions	22
1.4 Notions from Complexity Theory and the Efficiency of Algorithms	24
1.5 A Summary and Guide to the Thesis	28
CHAPTER 2: COMPLEXITY OF FLOWSHOP SCHEDULING WITH LIMITED TEMPORARY STORAGE	33
2.1 The Two Machine Finite Buffer Problem - (2,b)-FS	33
2.2 The Fixed Machine "No-Wait" Problem - (m,0)-FS	41
2.3 Three Machine Flowshops	56
CHAPTER 3: AN APPROXIMATION ALGORITHM FOR TWO MACHINE FINITE BUFFER FLOWSHOPS	62
3.1 The (2,0)-FS Problem and the Gilmore Gomory Algorithm	62
3.2 An Approximation Algorithm	66
3.3 Bounds for Flowshop Scheduling	73
3.4 The Significance of Permutation Scheduling	78
CHAPTER 4: THE ASYMMETRIC TRAVELING SALESMAN PROBLEM	88
4.1 Neighborhood Search Techniques and Primary Changes	88
4.2 The Significance of the Triangle Inequality	98
4.3 A Proposed Heuristic	102

TABLE OF CONTENTS (con't)

	Page
CHAPTER 5: CONCLUSIONS AND OPEN PROBLEMS	108
APPENDIX	110
REFERENCES	119

1. INTRODUCTION AND PROBLEM DESCRIPTION

1.1 Motivation

"What do I do first?" This irritating question has tormented us all. It has plagued the student in his thesis, vexed the engineer in his design, and worried the technician in his workshop. It is bound to puzzle any machine that aspires to be versatile and efficient. The automatic factory will have to live with it and above all answer it. What is most annoying is that it is never alone, but has a demanding sequel "What do I do next?"

In very general terms, that is the fundamental question the theory of scheduling set out to answer a quarter of a century ago. Because of its applicability to diverse physical systems as jobshops or computers, this theory spans many powerful disciplines such as Operations Research and Computer Science. Therefore it is not by chance that the ideas used in the present work were provided by the theory of Algorithms and Complexity.

Scheduling follows such functions as goal setting, decision making and design in the industrial environment. Its prime concern is with operating a set of resources in a time- or more generally cost-wise efficient manner. The present work was motivated by a number of obvious but significant facts about the operation of highly automated shops with multipurpose machines for batch manufacturing (e.g., a jobshop with computer controlled NC machines and an automated materials handling system). Let us briefly look at these facts:

(a) A large percentage of the decisions made by the supervisor or the computer control in such systems are scheduling decisions. Their objective is high machine utilization or equivalently schedules with as little idle time and overhead set-up times as possible. Therefore any analytic means to decide "what to do next?" with a guarantee of good performance will be welcome.

(b) Well behaved mathematical models (e.g., queueing models or network flow models) tend to appear at aggregate levels. Yet scheduling decisions are combinatorial in nature. A combinatorial approach can be useful both in the insight it provides and the determination of powerful non-obvious decision rules for giving priorities to different jobs. By making simplifying assumptions (e.g., replacing a complicated transfer line with a fixed size buffer) we can sometimes get a tractable combinatorial model.

(c) The trend in the hardware design is towards few powerful machines (typically 2 to 6) set in straight line or loop configurations with storage space and a transfer system. What is significant is the small number of machines, usually small buffer sizes (1 or 2 jobs in metal cutting applications) and the simple interconnection. The configuration studied consists of the machines in series (flowshop) under the restriction of finite buffers between machines (Figure 1). This does not give us the full job shop freedom of the transfer line as in (Figure 2). Yet let us look at an abstraction of the system in Figure 2. Consider a series of machines with buffers in between (which model the actual storage and the storage on the transfer line). The jobs flow through the machines and can also go from one buffer to the next

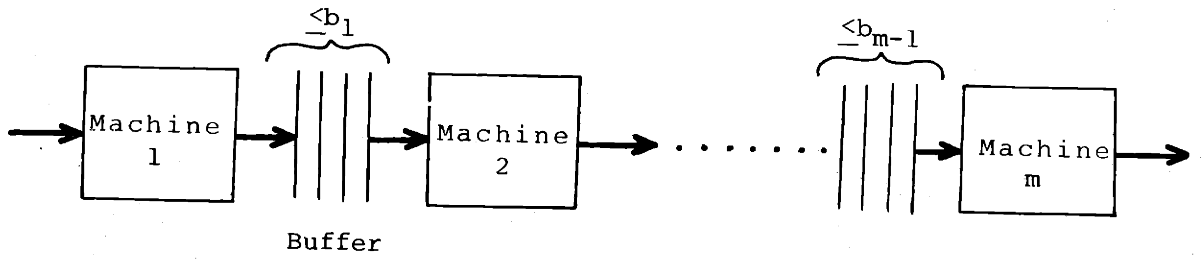


Figure 1

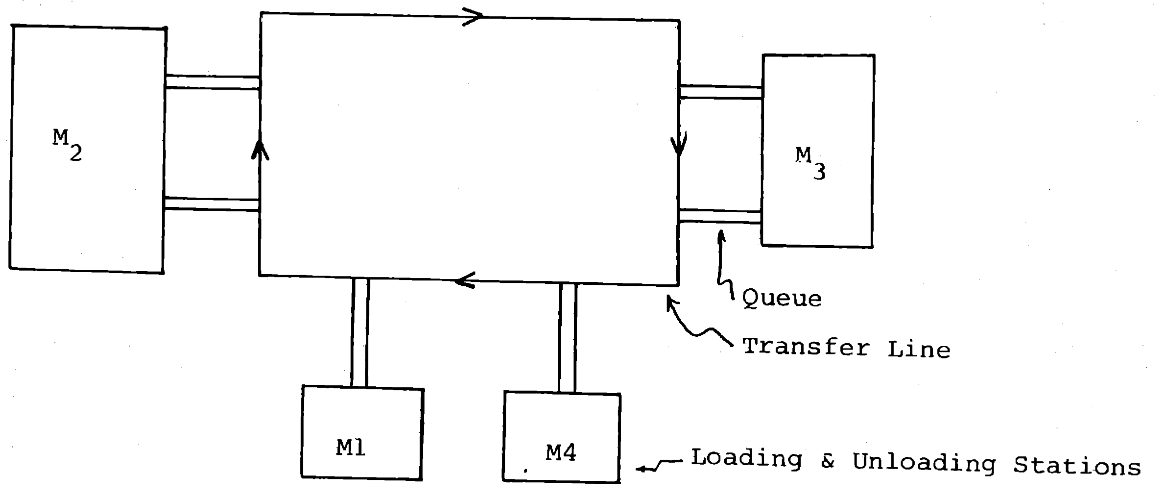


Figure 2

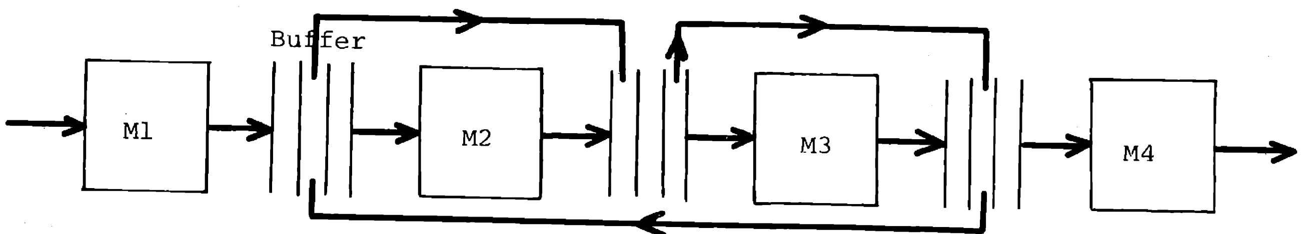


Figure 3

(Figure 3). Decidedly Figure 3, because of communication between buffers, is more complicated than Figure 1, which is a necessary first approximation.

(d) Scheduling models might sometimes look too simple, also the jobs are not always available when we determine the schedule. All this can be compensated by the fact that computers are much faster than other machines and we can experiment with simulation and heuristics (that the theory provides) long enough and often enough to make good decisions.

Let us now proceed with a detailed description of our scheduling problem, its setting in the context of the general scheduling model, the central ideas used from the theory of Algorithms and a brief look at the results obtained.

1.2 Problem Description

1.2.1 Discussion

We examine the problem of "scheduling flowshops with a limited amount of intermediate storage buffers".

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.

Two are the significant cases studied here:

(i) 2-machine flowshops with finite intermediate storage of size $b \geq 1$.

(ii) m-machine flowshops with no storage at all, where m is a fixed number larger than two; (also known as "no-wait" or "hot-ingot" problem.)

Apart from their practical significance, for which we argued in section 1.1, these problems are important because they are the simplest in the hierarchy of flowshop problems (see Figure 10), whose complexity the goal of this work was to study. Particularly problem (ii) can be formulated as an Asymmetric Traveling Salesman Problem (ATSP - given a directed graph and a weight function on the edges, find a path visiting all vertices exactly once at minimum cost). Let us now formulate our problems in a more rigorous manner.

1.2.2 The Finite Buffer Problem

We start by introducing (i). 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

---

*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 interger -- say n -- and then replacing 0 execution times by 1.

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. 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 4 it is demonstrated that this is different from having an extra buffer.

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

More formallly,

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 $S:\{1,\ldots,n\} \times \{1,2\} \to N$; $S(i,j)$ 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) $i \neq j \to S(i,k) \neq S(j,k)$.

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

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

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

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

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 4. Here 5 jobs are scheduled on two machines for different values of b, π is the identity permutation. In 4a and 4c a job leaves the first machine when it finishes, whereas in 4b and 4d it might wait. The buffers are used for temporary storage of jobs (e.g., job (3) in 4c 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.

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 [20]. Notice that, for $m > 2$, the m-machine, unlimited buffer problem is known to be NP-complete** [13]. Also for two machines, when no buffer space is available ($b = 0$, the "no-wait" case) the problem can be

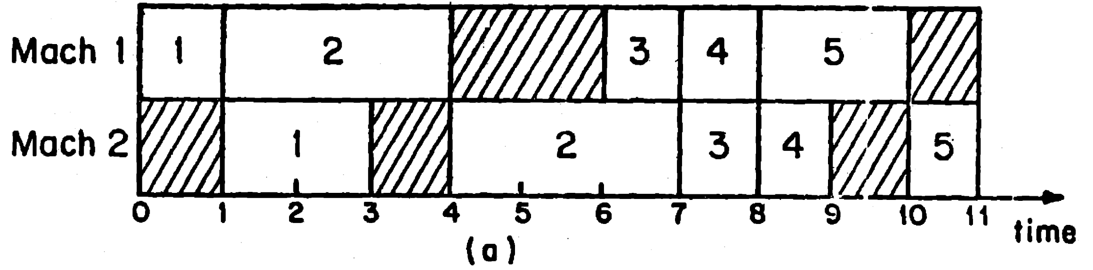
---

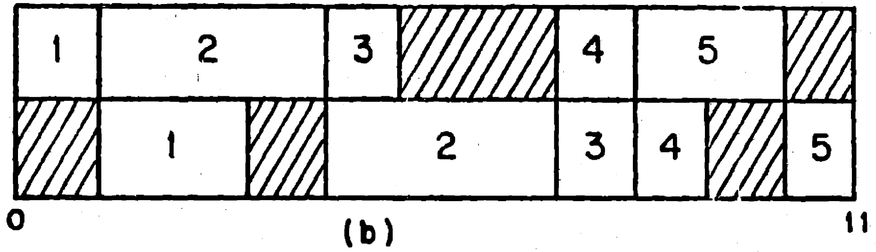
* If we assume that the first machine acts as temporary storage the only modification is that e) becomes

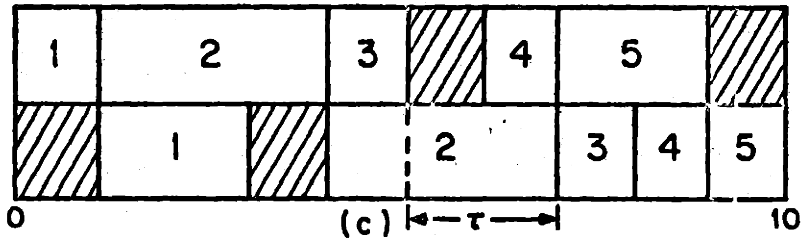
$$i > b+2 \rightarrow F(\pi(i-b-2),\ 2) \leq S(\pi(i),\ 1)$$

** See Section 1.4, characterizing the problem as NP-complete means that it is essentially computationally intractable.

i	a_i	β_i
1	1	2
2	3	3
3	1	1
4	1	1
5	2	1







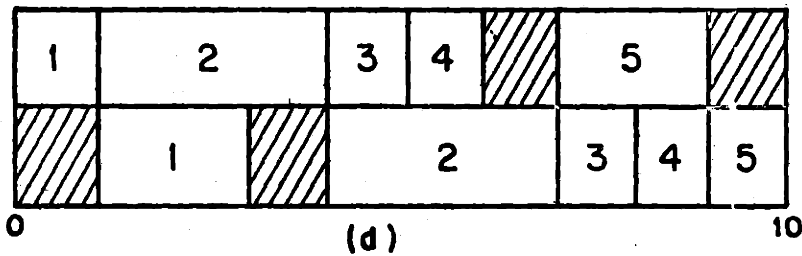


Figure 4

a) b=0

b) b=0

c) b=1

d) b=1

considered as a single-state machine problem in the fashion of [10]. As noted by [11], the case of the 2-machine flowshop problem in which b is given positive, finite integer was not as well understood. In fact, in [9] this practical problem is examined, and solutions based on dynamic programming are proposed and tested.

Although there are efficient algorithms for the extreme cases of zero and infinite buffer capacities, all the intermediate cases (finite capacity cases) are NP-complete. We analyze an efficient heuristic for solving the finite capacity problem.

1.2.3 The "No-Wait" Problem

Let us, now take a look at problem (ii) of 1.2.1. 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 [10] and such "no-wait" problems in general can be viewed as specially structured Traveling Salesman problems [32], [33], [39]. Furthermore, it was known that the problem is hard when m is allowed to vary as a parameter [24]. For fixed m and particularly $m = 3$ the complexity of the problem was an open question [24], [18]. 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.

We will demonstrate the connection between Asymmetric Traveling Saleman Problem (ATSP) and the scheduling problem by presenting the formulation of the "no wait" problem as a directed graph problem [33], [39]. We will comment on how the triangle inequality enters naturally

in this representation.

Each job is an m-tuple of numbers. We have the same restrictions of flowshop scheduling as in 1.2.2, only now b = 0 and $m \geq 2$. Let t_{ij} be the time to process job J_j on Machine M_i. Each job will correspond to a node in the graph. (We also have a start node with $t_{i0} = 0$ $1 \leq i \leq m$). The weights on the arcs (i,j) depend on the idle time on machine k, $1 \leq k \leq m$, that can result if job J_i precedes J_j in the schedule. Let us call this idle time $I_k(i,j)$. We have [33]:

$$I_1(i,j) = \max_k \left(\sum_{\tau=2}^{k} t_{\tau,i} - \sum_{\tau=1}^{k-1} t_{\tau,j},\ 0 \right) \qquad 2 \leq k \leq m \quad i \neq j$$

$$I_k(i,j) = I_1(i,j) + \sum_{\tau=1}^{k-1} t_{\tau,j} - \sum_{\tau=2}^{k} t_{\tau,i} \qquad 2 \leq k \leq m \quad 0 \leq i,j \leq n$$

There are k equivalent formulations as ATSP's with the distances expressing the idle times directly. Note that we have n+1 nodes, where the n nodes correspond to the n jobs and the n+1st one to the starting node with all processing times 0.

$$ATSP_k \; : \; C_{ij}^k = I_k(i,j) \qquad \begin{array}{c} 1 \leq k \leq m \\ 0 \leq i,j \leq n \end{array}$$

If for simplicity the optimal permutation of jobs is the identity we have:

$$(\text{Optimal Schedule}) = \underbrace{\sum_{\tau=1}^{n} t_{k\tau}}_{\text{Total Processing on Machine } k} + (\text{Optimal tour of } ATSP_k) =$$

$$= \sum_{\tau=0}^{n} t_{k\tau} + c_{01}^{k} + \sum_{\tau=1}^{n-1} c_{ii+1}^{k} + c_{n0}^{k} =$$

$$= \sum_{i=0}^{n} (c_{i(i+1)\bmod(n+1)}^{k} + t_{ki}) = \sum_{i=0}^{n} d_{i(i+1)\bmod(n+1)}^{k} = \qquad (*)$$

$$= \sum_{i=0}^{n} (c_{i(i+1)\bmod(n+1)}^{k} + t_{k(i+1)\bmod(n+1)}) = \sum_{i=0}^{n} p_{i(i+1)\bmod(n+1)}^{k} \qquad (**)$$

where $d_{ij}^{k} = c_{ij}^{k} + t_{ki}$ $\qquad p_{ij}^{k} = c_{ij}^{k} + t_{kj}$.

The TSP formulations based on the d_{ij}^{k}'s and the p_{ij}^{k}'s satisfy the triangle inequality.

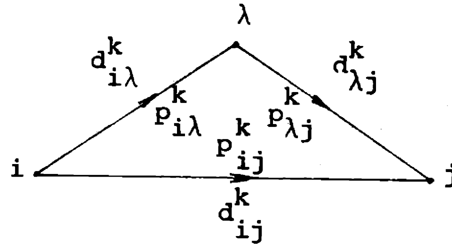


Figure 5

$d_{i\lambda}^{k} + d_{\lambda j}^{k} \geq d_{ij}^{k}$ and $p_{i\lambda}^{k} + p_{\lambda j}^{k} \geq p_{ij}^{k}$ because:

$$d_{i\lambda}^{k} + d_{\lambda j}^{k} - d_{ij}^{k} = I_k(i,\lambda) + I_k(\lambda,j) - I_k(i,j) + t_{ki} + t_{k\lambda} - t_{ki} =$$

$$= p_{i\lambda}^{k} + p_{\lambda j}^{k} - p_{ij}^{k} = I_1(i,\lambda) + I_1(\lambda,j) - I_1(i,j) + t_{1\lambda} = \text{(by substitution)}$$

$$= \sum_{\tau=1}^{K_1} t_{\tau i} - \sum_{\tau=1}^{K_1-1} t_{\tau\lambda} + \sum_{\tau=1}^{K_2} t_{\tau\lambda} - \sum_{\tau=1}^{K_2-1} t_{\tau j} - \left(\sum_{\tau=1}^{K_3} t_{\tau i} - \sum_{\tau=1}^{K_3-1} t_{\tau j}\right) \geq$$

(by adding $t_{1i} - t_{1i}$ and substituting)

$$\geq \left(\sum_{\tau=1}^{K_3} t_{\tau i} - \sum_{\tau=1}^{K_3-1} t_{\tau\lambda}\right) + \left(\sum_{\tau=1}^{K_3} t_{\tau\lambda} - \sum_{\tau=1}^{K_3-1} t_{\tau j}\right) - \left(\sum_{\tau=1}^{K_3} t_{\tau i} - \sum_{\tau=1}^{K_3-1} t_{\tau j}\right) \geq 0$$

The formulation commonly used is with

$$d^1_{ij} = \max_{1\leq k\leq m} \left(\sum_{\tau=1}^{k} t_{\tau i} - \sum_{\tau=1}^{k-1} t_{\tau j} \right)$$

Example: In order to illustrate the above let us formulate the $ATSP_1$ for 2 machines and jobs $(t_{1i}, t_{2i}) = (\alpha_i, \beta_i)$

$$C_{ij} = \max (\beta_i - \alpha_j, 0)$$

$$d_{ij} = \max (\alpha_i + \beta_i - \alpha_j, \alpha_i)$$

$$P_{ij} = \max (\alpha_j, \beta_i)$$

For the $ATSP_1$ for 3 machines and jobs $(\alpha_i, \beta_i, \gamma_i)$ we have:

$$C_{ij} = \max (\beta_i + \gamma_i - (\alpha_j + \beta_j), \beta_i - \alpha_j , 0)$$

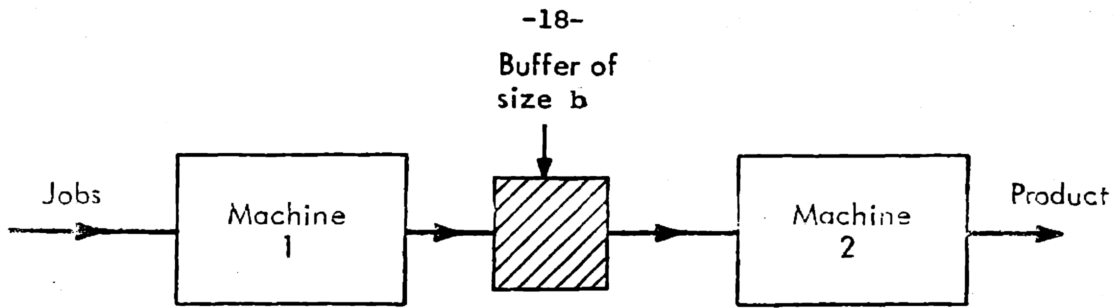
$$d_{ij} = \max (\alpha_i + \beta_i + \gamma_i - (\alpha_j + \beta_j), \alpha_i + \beta_i - \alpha_j, \alpha_i)$$

$$P_{ij} = \max (\beta_i + \gamma_i - \beta_j, \alpha_j, \beta_i)$$

Note that in the first example all distances P_{ij} are increased by an increase in the job lengths. In the second example an increase in the β's will increase some distances and might decrease others.

For the set of tasks $\mathcal{J}$ figure 6 has the $ATSP_2$ formulation and figure 7 the Ghannt charts for schedules using 1 or 0 buffers.

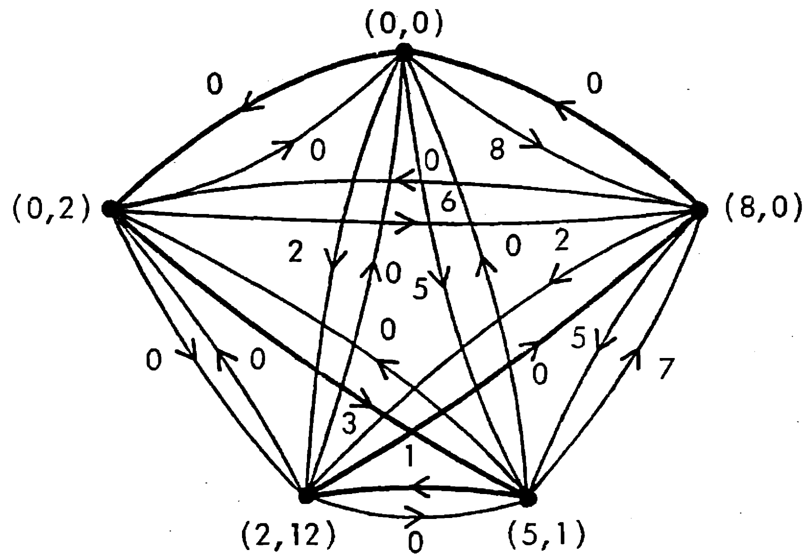
Certain remarks are significant. First if we assume a formulation using idle times as distances and consider only the arcs with 0 weight, the existence of a Hamilton circuit (a circuit visiting all nodes exactly once) in the resulting graph indicates that there is a schedule saturating machine k. Second we actually have mn degrees



(a) Two-machine flowshop with buffer size b

Job_i	Time spent on Machine 1 (t_{1i})	Time spent on Machine 2 (t_{2i})
1	≈0	2
2	2	12
3	5	1
4	8	≈0

(b) A set of tasks $\mathcal{T}$

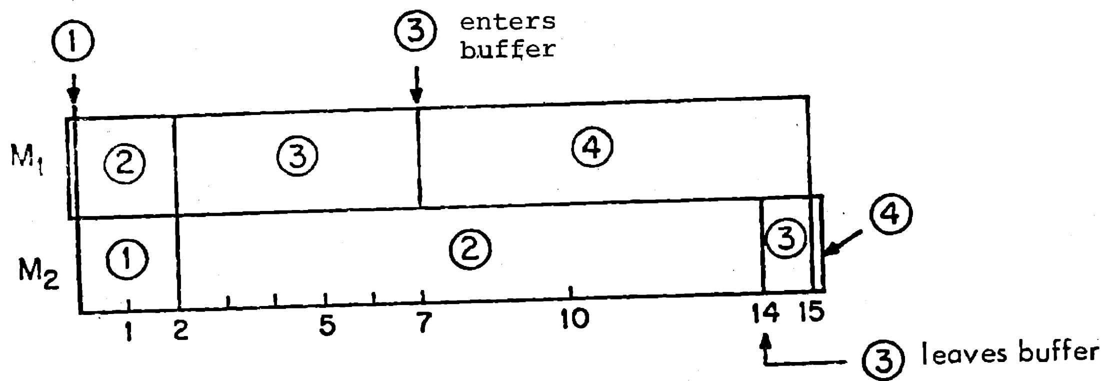


(c) The TSP, which corresponds to $\mathcal{T}$ and $b = 0$. Optimal tour is in heavy lines. The distances of the TSP are C_{ij}

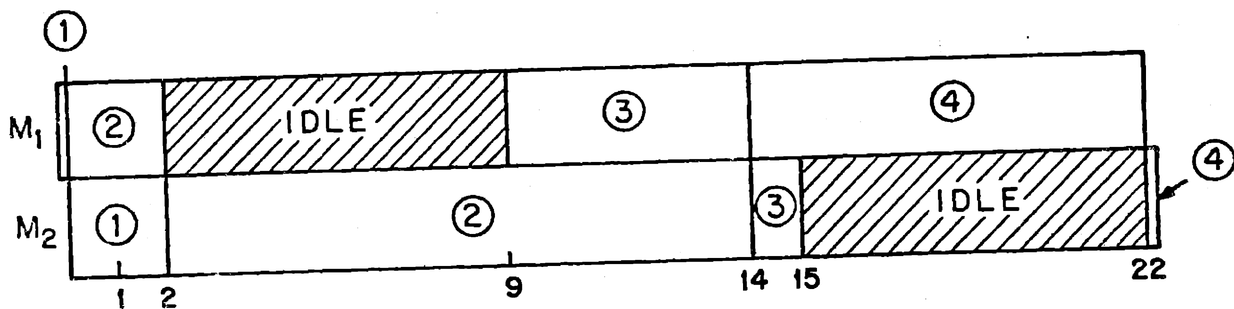
C_{ij} is the idle time on Machine 2, if job j follows job i

$$C_{ij} = \max\left\{0,\ t_{1j} - t_{2i}\right\} \quad C_{io} = 0$$
$$C_{oj} = t_{1j}$$

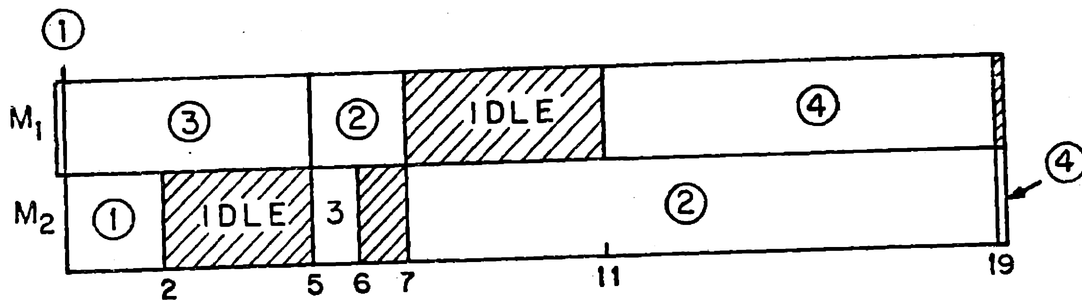
Figure 6



(a) The optimal schedule for $\mathcal{J}$ if b = 1. The permutation of jobs is ① - ② - ③ - ④



(b) The schedule for the permutation of jobs ① - ② - ③ - ④ and b = 0. [(a) is a compression of (b)]



(c) The optimal schedule for $\mathcal{J}$ if b = 0. The permutation of jobs is ① - ③ - ② - ④

Figure 7

of freedom to construct a matrix of $(n+1)n$ distances which satisfy the triangle inequality. If m is a parameter we can take $m=O(n^2)$ and expect to come up with a problem as difficult as the general TSP. This intuition is correct [24]. If m is fixed the problem of determining its complexity is more difficult.

1.2.4 The Significance of Constraints

Let us briefly comment on three issues related to the constraints in our problem definitions.

1) If we assume that a machine can serve as temporary storage this does not affect our analysis (e.g. increase b), and produces only minor modifications in the schedule (see figure 4).

2) By accepting jobs of 0 length we do not assume that they can bypass the machines or leave the system. This is in contrast with [36], which treats the complexity of "no-wait" job-shops and open-shops. Also [36] introduces a "flow shop with 0 tasks". Such a difference in definition (jobs able to bypass machines) makes it simple to derive NP-completeness results from partition problems, but the problem is no longer the "no-wait" fixed size flowshop problem [24].

3) In a flowshop of m machines and n jobs the combinatorial optimization problem addresses the question of determining a permutation of jobs to be performed on each machine (that is m permutations π_i $1\leq i\leq m$, such as those in the definition of 1.2.2). Therefore the space of permutations that we must search is roughly $O((n!)^m)$. We would like to be able to search a smaller space. For all buffers ∞

there are two classical results in scheduling theory [7], which demonstrate that we can restrict our attention to equal permutations on the first two machines and the two last ones (for a makespan criterion), without loss of optimality. Since the problem is difficult anyway it is customary to look only for some good schedule with the same permutation of jobs for all machines (optimal in the space of permutation schedules). This is obviously the correct approach when all buffers are 0, yet it does not hold for b finite > 0. Even when b=1, m=2 we might have a job bypassing another job that remains in the buffer and allowing $\pi_1 \neq \pi_2$ our schedule might fare slightly better than under the FIFO assumption. This is demonstrated in Figure 8 and more closely examined in chapter 3. 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

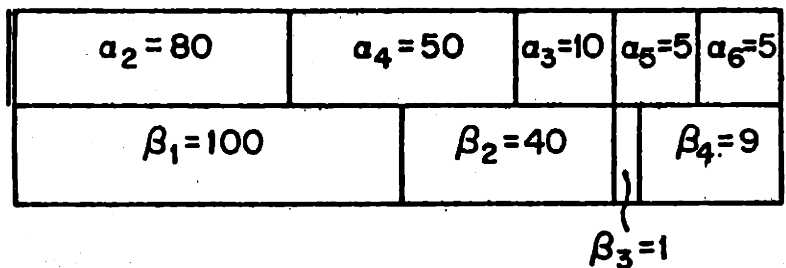


Figure 8

b=1 $\pi_1=\pi_2$

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

1.3 The General Scheduling Model & Extensions

In this section we will sketch the simple combinatorial model, which defines most deterministic scheduling problems. This is done in order to provide a complete view of the setting in which the present work fits. Also this section provides us with a common language for problems addressed in chapters 2 and 5 or referred to elsewhere in this thesis.

Suppose that n jobs $J_j (j=1,...,n)$ have to be processed on m machines $M_i (i=1,...,m)$. The most basic assumptions are that each machine can process at most one job at a time and that each job can be processed on at most one machine at a time. The scheduling problem has three dimensions, which are more thoroughly examined in [18].

1) Resources: We have m machines, where m might be fixed or a problem parameter. We might have a single machine, identical, uniform (they perform the same tasks at different speeds) or unrelated parallel machines. Common network configurations are the flow-shop (defined in 1.2), the open-shop (each J_j consists of a set of tasks one for each of the m machines, but the order of execution is immaterial) and the job-shop (each J_j consists of a chain of tasks on a subset of the machines). In general we also have a set of resources, which can be shared and which are scarce

$$R = \{R_1,...,R_s\} \text{ where } R_i \leq B_i \qquad 1 \leq i \leq s .$$

The buffer resources examined in this work can both be shared and form part of the network topology of the system. Generally there seems to be a jump in complexity when going from two- to three-machine systems.

2) The Job Characteristics

The jobs can be characterized by

a) Precedence constraints expressed as an irreflexive partial order on the tasks. This partial order $<$ is represented by a directed acyclic graph, which in special cases can have the form of a tree or a set of chains (as in the flowshop case).

b) Processing times $t_{ij} \geq 0$ of J_j on M_i.

c) Resource requirements $R_i(J_j) \leq B_i$ of a job J_j for resource i (B_i = maximum available resource).

d) Release dates r_j, on which J_j becomes available and due dates d_j, by which J_j should ideally be completed. If $F_i(S)$ denotes the finishing time of a job in a schedule S, then $F_i(S)-d_i$ is its lateness and $\max\{0, F_i(S)-d_i\}$ its tardiness.

3) Optimization Criteria

Our goal is to determine a sequence of tasks S such that certain performance measures are optimized. These optimal schedules might belong to general classes: preemptive, nonpreemptive (each task is an entity and cannot be subdivided and executed separately), list scheduling (a priority list is constructed and tasks are taken from it if the proper processor is free and the precedence constraints satisfied), permutation scheduling (in flow shops as discussed in 1.2.4) etc. The common performance measures are maximum finishing time $\max_{1 \leq i \leq n} (F_i(S))$ or makespan, mean weighted finishing time, maximum or mean weighted lateness and tardiness.

In the last few years we have witnessed a spectacular progress towards understanding deterministic multiprocessor scheduling problems of various types. For an overview of results in scheduling we recommend [5]; [11], [18] and [7] also stress certain aspects of the area.

1.4 Notions from Complexity Theory and the Efficiency of Algorithms

In order to investigate the computational complexity of a problem, the meanings of "problem" and of "computation" must be formally defined. We will attempt to illustrate these definitions.

A problem might be considered as a question that has a "yes" or "no" answer. Normally the question has several parameters, i.e., free variables. A selection of values for the parameters is termed an instance of the problem. The size of the instance is the length of the string used to represent the values of the parameters. When we speak about complexity $O(n^3)$, n is the size of the instance.

The time needed by an algorithm expressed as a function of the size of the instance of the problem is called the time complexity of the algorithm. The limiting behavior of the complexity as size increases is called the asymptotic time complexity. Analogous definitions can be made for space complexity and asymptotic space complexity.

The theory of computation has created and analyzed simple computer models (automata) in order to determine the limits of computer capabilities and make precise the notion of an algorithm. This is where the study of time and space requirements on computation models (e.g., Turing machines, pushdown automata, etc.) has been fruitful. A distinct line has been drawn between easy and hard problems. Easy problems are determined

as those solvable efficiently or those that have polynomial time complexity. These easy problems belong to the class P. The other interesting class for our purposes is NP (for nondeterministic, polynomial time). It encompasses P and all problems, which can be solved by a nondeterministic Turing machine in polynomial time.

All problems in NP are solvable in principle, but for those whose status with respect to P is still uncertain only exponential time algorithms are known. So in order for a problem to qualify for the class NP there need not be an efficient means of answering the yes-or-no decision questions. What polynomial time on a nondetermistic Turing machine in fact says is that the problem is solvable by an algorithm with the ability to backtrack, which would take polynomial time if a wise demon made the correct choice at every point in our program and the final answer were yes. In other words what is required is that whenever the answer is yes, there be a short and convincing argument proving it.

The structure of NP has been closely examined but we need only consider one of its subsets, the subset of hard problems. Although some theoretical questions have still to be resolved, we can safely state the hard problems will be the NP-complete ones.

<u>Definition</u>: Define problem (S) to be NP-complete if every problem in NP polynomially reduces to (S). An important corollary is that if a polynomial-time algorithm can be found for problem (S), then polynomial algorithms will be determined for all problems in NP.

The key theorem for NP-complete problems was given by Cook [8]. Many important NP-complete problems were investigated by Karp [21] and

an up to date bibliography is contained in [15].

The key problem is the satisfiability problem for Boolean expressions, which, given a Boolean expression, is as follows:

> "Does there exist an assignment of truth values 0 and 1 (false & true) to its propositional variables that makes the expression true?"

Cook proved that:

Satisfiability problem ϵ NP-time
if (s) ϵ NP-time then (s) reduces polynomially to the satisfiability problem therefore
satisfiability problem is NP-complete

Therefore in order to prove NP-completeness for other problems all we need are polynomial transformations from known NP-complete problems -- e.g., the satisfiability problem, the clique problem (is there a complete subgraph in a graph), etc. -- to the problems we wish to prove NP-complete. Since these problems are in NP, Cook's result stated above actually closes the circle and proves complete equivalence between NP-complete problems, justifying the fact that they are the hardest in NP.

Ullman [37] initially investigated NP-completeness of scheduling problems and some results exist on the subject to date, i.e., [12], [13], [15], which are well classified by Lenstra [24]. Let us dwell now on the practical significance of these results. Fig. 9 is the standard representation of NP (co-NP are the complementary decision problems of NP).

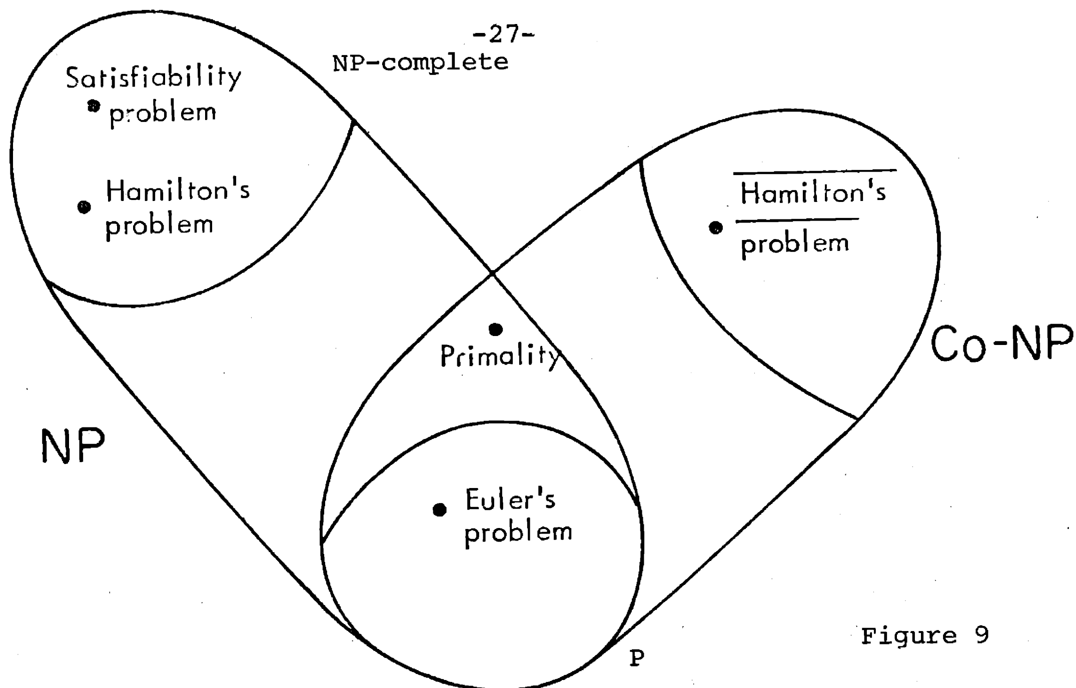


Figure 9

Some general remarks may be made concerning NP-completeness:

a) Proving NP-completeness is, in a way, a negative result, because then only general-purpose methods like dynamic programming, integer programming or branch and bound techniques could give an exact solution. Yet this study provides the motivation and the proper problems for refining such techniques (based on such general methodologies good approximate solutions are always welcome).

b) Proofs of NP-completeness are significant for problems on the borderline between hard and easy problems. These proofs are generally difficult (there is no standard way to come up with them) and the problems that still remain open are candidates for a breakthrough in research, or might belong to a separate complexity class. The problem reductions unify the field of combinatorial optimization.

The obvious question is <u>"what can we do about NP-complete problems?"</u> There are various possible approaches:

1) Application of general purpose methods and heuristics based on our problem's properties, where experience actually gives us the

certificate of "good" behavior of our algorithm (such an approach will be described in Chapter 4).

2) Deterministic approximation algorithms, which run in polynomial time and for which we can derive analytic upper bounds, exhibit worst case examples and determine special solvable cases for our problem (Chapter 3 is devoted to this methodology).

3) Average case analysis of an algorithm, which is equivalent to a randomization of our inputs. We either use analytic means or simulation (Chapter 3).

4) By extending the ideas of randomization we can examine probabilistic algorithms [22].

In the following paragraph we will discuss how the problems described in 1.2 were examined and treated using the ideas above.

1.5 A Summary and Guide to the Thesis

The problems addressed in this thesis are those formulated in 1.2. We will use the following abreviations for them.

- For the 2 machine flowshop problem with finite intermediate storage size $b \geq 1$ we use (2,b)-FS.
- For the m-machine, (m fixed), "no-wait" flowshop problem we use (m,0)-FS (e.g., m=4 (4,0)-FS).

Chapter 2 contains a study of the complexity of these problems. Chapter 3 an approximation algorithm for the (2,b)-FS and Chapter 4 a scheme for a heuristic approach to the (m,0)-FS problem. Chapter 5 contains our conclusions and a general discussion, whereas an Appendix

contains the LISP listing for a program used in the (2,b)-FS approximation.

The research was conducted in close collaboration with Prof. C.H. Papadimitriou of Harvard, whose great help and interest are enormously appreciated. Most of the basic results have appeared in [31]. Let us briefly describe this research.

a) A certain number of open problems in the borderline between easy and hard problems have been proven NP-complete. The <u>(2,b)-FS problem</u> has been proven NP-complete by two different reductions for $b \geq 3$ and $b \geq 2$ using a partition problem. Using a more restricted partition problem Prof. C.H. Papadimitriou has proven NP-completeness for the $b \geq 1$ case. Other problems examined are three machine flowshops with buffer sizes of 0 and ∞ or ∞ and 0 respectively. The <u>(m,0)-FS problem has been proven NP-complete for $m \geq 4$.</u> The proof technique is based on [16]; which actually creates gadgets which translate Boolean algebra into statements about Hamilton paths. This part represents a joint effort with Prof. C.H. Papadimitriou. Personal contributions are the idea of use of Hamilton paths in the defined special graphs and the realization of the components of these graphs in terms of 4 machine job systems. This (m,0)-FS problem was pointed out by Lenstra [24]. As far as the no-wait problems are concerned we leave only one open question: the 3-machine case (i.e., (3,0)-FS)). 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.

These are the contents of Chapter 2, Figure 10 contains the hierarchy of these flowshop problems in a schematic manner.

b. For the (2,b)-FS problem we have an obvious heuristic.

STEP 1 Solve the TSP problem resulting from a constraint of 0 sized buffers

STEP 2 Compress the resulting schedule to take advantage of finite buffer size.

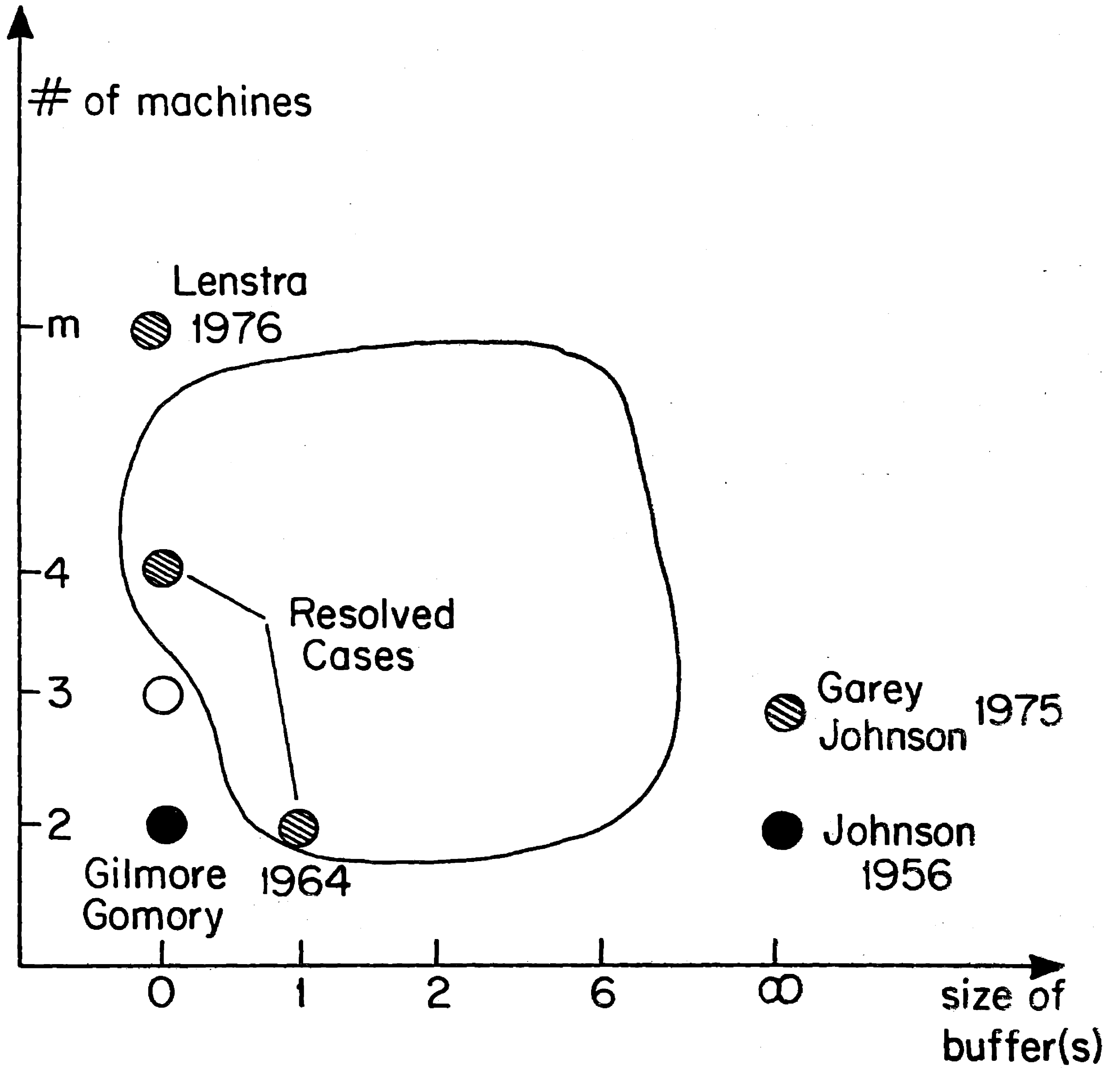
STEP 1 is the Gilmore Gomory algorithm [10], which is reviewed and implmented in $0(n\log n)$ time (which is optimal) as opposed to $0(n^2)$. Let $\mu_b(J)$ denote the minimal schedule length for a job system in a (2,b)-Flowshop. Prof. C.H. Papadimitriou has proven that $\frac{\mu_b(J)}{\mu_0(J)} \leq \frac{2b+1}{b+1}$. This gives on upper bound on the accuracy of the approximation of $\frac{b}{b+1}$ We will outline the argument for b=1. Worst case examples indicate that the bound is tight even after STEP2. The only other exact bounds for flowshops are [3] and [17].

Since it was proven that for every job set we can find another job set with a saturated optimal schedule and worse performance for the above heuristic than the initial job set, we tested the average case performance of the heuristic with respect to such saturated optimal schedules. The performance was very encouraging (average error around 5% and deviation around 10% for large number of jobs).

Finally, we examine the effect of having FIFO buffers that cannot be bypassed by jobs (let $\bar{\mu}_b(J)$ be the optimal makespan with b general buffers). We provide arguments to substantiate the conjecture that $\frac{\mu_1(J)}{\bar{\mu}_1(J)} \leq \frac{5}{4}$. These are the contents of Chapter 3.

c) All the above flowshop problems actually are or can be approximated by special cases of the Asymmetric Traveling Saleman problem. Although branch and bound techniques have been studied for this problem [24], heuristics equivalent to the most successful Lin Kemighan algorithm [25] have not been developed. The fact that the triangle inequality enters naturally in these applications (see section 1.2.3) makes it plausible (Theorem 4.2.1) that heuristics based on neighborhood search will perform well. What is non obvious is how to extend the Lin-Kernighan methodology from undirected to directed graphs. What we attempt to do is study primary changes [38] and the neighborhood structure resulting from them in the asymmetric case (e.g., all primary changes in the ATSP are odd). Based on this we propose a scheme for the extension of the Lin Kernigham algorithm to the asymmetric case. These are the contents of Chapter 4.

Hoping that we have appropriately motivated and decribed the contents of this research, we will now proceed to the actual theorems and proofs.



PROBLEM: FLOWSHOP SCHEDULING WITH LIMITED TEMPORARY STORAGE IN ORDER TO MINIMIZE SCHEDULE LENGTH

- ● easy problems
- ◍ hard problems
- ○ open problems

Figure 10

2. COMPLEXITY OF FLOWSHOP SCHEDULING WITH LIMITED TEMPORARY STORAGE

2.1 The Two Machine Finite Buffer Problem - (2,b)-FS

We will prove that the two machine fixed, non-zero, finite buffer flowshop problem ($b \geq 1$) is NP-complete. We use the shorthand notation (2,b)-FS. This is somewhat surprising given that the limiting cases of $b=0$ and $b=\infty$ (or $b \geq n$) have easy (polynomial) solutions [10], [20].

The corresponding decision problem is:

Given n jobs and an integer L, is there a feasible schedule S for the (2,b)-FS such that $\mu(S) \leq L$? (b is a fixed non-zero integer).

We can use different reductions for different fixed sizes of b. The problems, which are efficiently reduced to the (2,b)-FS and known to be NP-complete are described in [15]. The technique the reductions use is similar to [13]. The basic idea behind the proofs is: given an arbitrary instance of a partition problem for integers we translate it into a job system for a flowshop which, iff the partition problem has a solution, will have a saturated (no-idle time) schedule. This job system is such as to create slots of known length in the Ghannt chart, which have to be filled in exactly if we wish to meet the schedule length L requirement.

We will use two partition problems designated 3M and 3MI (for three way matching of integers, where 3MI is a special (restricted version) of 3M).

Three-way matching of integers (3M)

Given a set A of 3n integers $A = \{a_1, \ldots, a_{3n}\}$ with $\sum_{i=1}^{3n} a_i = nc$ does there exist a partition of A into three element sets A_i $1 \leq i \leq n$, such that for each i $\sum_{a_j \in A_i} a_j = c$?

Without loss of generality $\frac{c}{4} < a_i < \frac{c}{2}$ and the problem is NP-complete [13].

Special 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 [15].

In the 3MI problem we wish, as in 3M, to partition a set $(A \cup B)$ into three element sets, but these three element sets contrain one element from A and two from B.

The (2,b)-FS problems are obviously in NP. A nondeterministic algorithm could guess the optimal permutation π, construct the corresponding schedule S, and check that $\mu(S) \leq L$. Let us now proceed with the reductions.

Theorem 2.1.1. The (2,3)-FS problem is NP-complete.

Proof: Suppose that we are given an instance $\{a_1, \ldots, a_{3n}\}$ of the 3M problem. We construct the following instance of the (2,3)-FS problem. Consider a set $\mathcal{J}$ of 4n+2 jobs, with execution times (α_i, β_i) as follows:

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

b) For each i, $1 \le i \le 3n$ we have a job $E_i = (0, a_i)$.
L is taken to be $(2n+1)c$.

"$\Rightarrow$" Suppose a partition exists which has the desired form. That is each set A_i consists of three elements $a_{g(i,1)}, a_{g(i,2)}, a_{g(i,3)}$ such that for all i, $1 \le i \le n$ $\sum_{j=1}^{3} a_{g(i,j)} = c$. Then the following schedule shown in Figure 11a and by its starting times below has length $(2n+1)c$.

$S(K_0, 1) = 0 \qquad S(K_0, 2) = 0$

$S(K_i, i) = 2c(i-1) \qquad S(K_i, 2) = 2ci$

$S(K_{n+1}, 1) = 2cn \qquad S(K_{n+1}, 2) = 2cn + c$

$S(E_{g(i,1)}, 1) = 2c(i-1) \qquad S(E_{g(i,1)}, 2) = 2c(i-1)$

$S(E_{g(i,2)}, 1) = 2c(i-1) \qquad S(E_{g(i,2)}, 2) = 2c(i-1) + a_{g(i,1)}$

$S(E_{g(i,3)}, 1) = 2c(i-1) \qquad S(E_{g(i,3)}, 2) = 2c(i-1) + a_{g(i,1)} + a_{g(i,2)}$

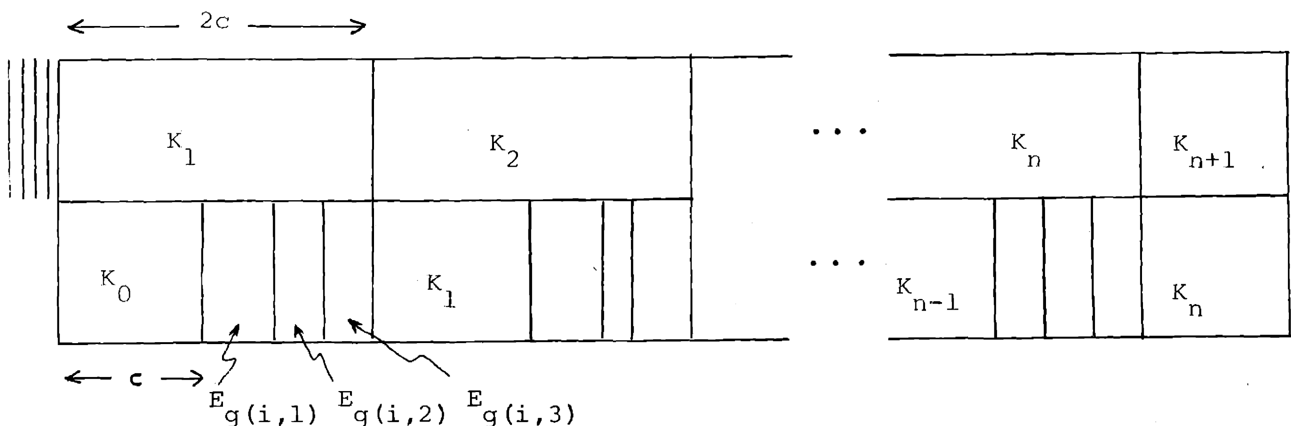


Figure 11a

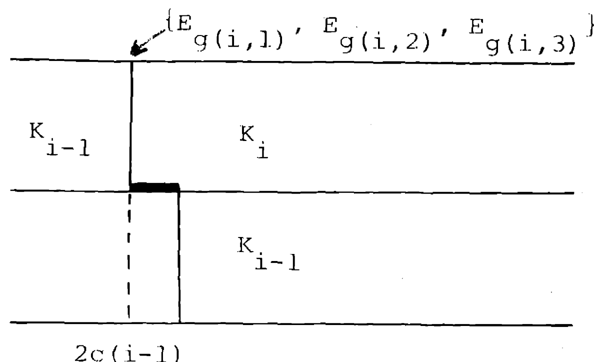


Figure 11b

"$\Leftarrow$" Conversely suppose a schedule of length $(2n+1)c$ exists. It is evident that there is no idle time on either machine. The schedule on machine 1 has the form $K_0, K_1, K_2, \ldots, K_n, K_{n+1}$ because all K_i's $1 \leq i \leq n$ are equivalent. The $3n$ jobs E_j have to be distributed among the n instants $2c(i-1)$, $1 \leq i \leq n$ for the first machine. We cannot insert 4 or more jobs at time $2c(i-1)$ on machine 1, because since K_i has to start on it no matter which one of the 4 jobs or K_{i-1} we executed on machine 2 we would violate the buffer constraint, (Figure 11b is not permitted). Therefore, since we have $3n$ jobs we have to place exactly 3 of them at each $2c(i-1)$. Betwee $2c(i-1)$ and $2ci$ machine 2 must execute exactly these 3 jobs and K_{i-1}. The schedule must be saturated thus the three E_j's allocated to $2c(i-1)$ must correspond to three a_i's that sum up to c.

Theorem 2.1.2 The (2,2)-FS problem is NP-complete.

Proof: Suppose we are given an instance $\{a_1, \ldots, a_{3n}\}$ of the 3M problem. We construct the following instance of the (2,2)-FS problem.

Consider a set J of $4n+2$ jobs with execution times (α_i, β_i) as follows:

a) We have n jobs $K_1, \ldots, K_n$ with $K_i = (c, \frac{3c}{16})$. Also, we have jobs $K_0 = \left(0, \frac{3c}{16}\right)$ and $K_{n+1} = (\frac{3c}{16}, 0)$.

b) For each $1 \leq i \leq 3n$ we have a job $E_i = (\frac{c}{16}, a_i)$.
L is taken to be $nc + (n+1)\frac{3c}{16}$

"$\Rightarrow$" Again if a partition exists we have a schedule as in Figure 12.

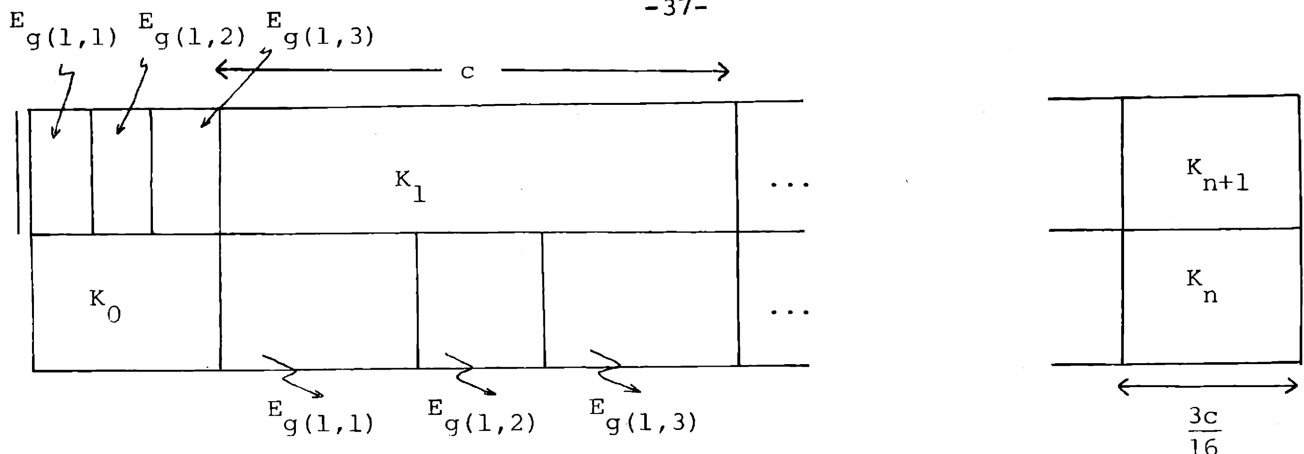


Figure 12

Conversely if there is a schedule it must be saturated. On machine 1 between K_{i-1} and K_i we cannot have more than three jobs, because the buffer would overflow. We now use the property $\frac{c}{4} < a_i < \frac{c}{2}$. We cannot have more than 3 E_j jobs consecutive, because the processing times on machine 2 are $\frac{3c}{16}$ or at least greater than $\frac{c}{4}$. The rest of the argument is as in theorem 2.1.1.

Theorem 2.1.3* The (2.1)-FS problem is NP-complete.

Proof: 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:

*This reduction is due to Prof. Ch.H. Papadimitriou

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 $A_i = (c/2,a_i)$.

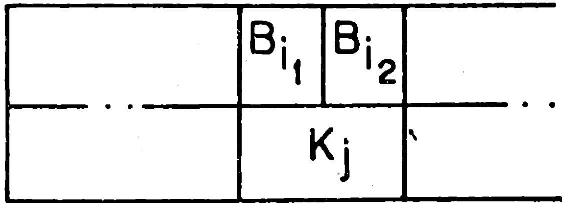
L is taken to be n(c+2); this complete the construction of the instance I of the (2,1)-FS.

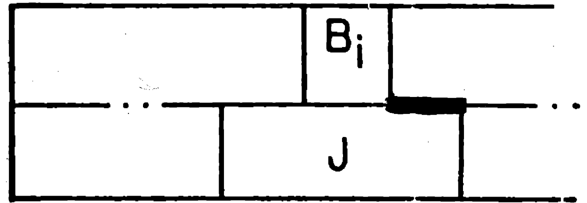
We shall show that I has a schedule S with $\mu(S) \leq L$ iff the original instance of the 3MI problem had a solution. First notice that L equals the sum of all α_i's and also of all β_i's; hence $\mu(S) \leq L$ iff $\mu(S) = L$ and there is no idle time for either machine in S. It follows that K_0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

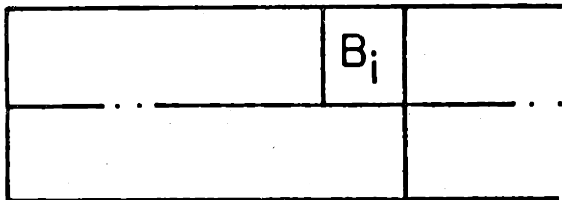
We shall need the following lemma:

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

<u>Proof of Lemma</u>. 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 13a. Obviously, the case shown in Figure 13b -- the execution of B_i on the first machine starts and end 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 13c. 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 13d is also shown impossible. Furthermore, identical arguments hold for subsequent executions of B_i jobs; the lemma follows.







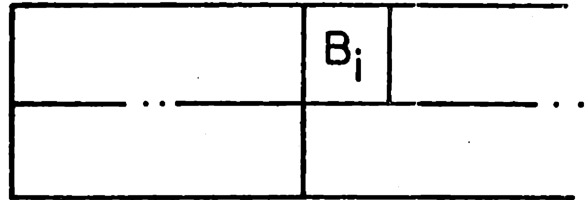


Figure 13

By the lemma, any schedule S of $\mathcal{J}$ 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 14). 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 $a_i + p_{i_1} + p_{i_2} = C$.

1	1	C/2	C/2	1	1	C/2	C/2
2	$b_{i_1} = P_{i_1}$	$b_{i_2} = P_{i_2}$	a_i	2	b'_{i_1}	b'_{i_2}	a'_i

Figure 14

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 14. Hence we have shown that the 3MI problem reduces to (2,1)-FS, and hence the (2,1)-FS problem is NP-complete.

Let us now notice that the above arguments, for $b \geq 1$, generalize to show that the (b+2)M problem reduces to the (2, b+1)-FS and that the (b+2)MI problem reduces to the (2,b)-FS. (e.g., 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 $a_i + \sum_{j=1}^{b+1} p_{i_j} = c$. This problem is easily seen to be NP-complete.) Therefore the important conclusion is that

<u>Corollary</u> The (2,b)-FS problem is NP-complete for any fixed b, $0<b<\infty$.

Let us close this section by commenting on the use of another optimization criterion. If instead of the makespan we wish to examine flowshops under a flowtime criterion we have that the (2,b)-FS is NP-complete for $b=\infty$ and open for b=0 [24]. We conjecture that the (2,1)-FS under a flow time criterion is NP-complete.

2.2 The Fixed Machine "No-Wait" Problem - (m,0)-FS

In certain applications we must schedule flowshops without using any intermediate storage; this is known as the <u>no-wait</u> problem. (For a discussion of this class of problems, see Section 1.2.3).

<u>Theorem 2.2.1</u>* The (4,0)-FS problem is NP-complete

---

*The discussion of this proof follows [31]. This part represents a joint effort with Prof. C.H. Papadimitriou. Personal contributions are the idea to use Hamilton paths in the defined special graphs and the realization of the components of these graphs in terms of 4 machine job systems.

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., $k \in K$ $F(i,k) = S(j,k)$).

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 15) 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 15

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 $K \subseteq \{1,2,\ldots m\}$

(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 2.2.1 by the following result.

Theorem 2.2.2 The $(4;\{2,3\})$-Hamilton circuit problem is NP-complete.

We shall prove Theorem 2.2.2 by employing a general technique for proving Hamilton path problems to be NP-complete first used by Garey, Johnson and Tarjan [16]. (See also [26], [27].) 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 16) 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 exactly one arc from each pair P.

b. C traverses at least one arc from each triple T.

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

Proof. We shall reduce 3-satisfiability to it. Given a formula F involing 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 17a) and a set of triples T (Figure 17b), such that D has a feasible -- with respect to P and T -- Hamilton circuit iff the formula is satisfiable.

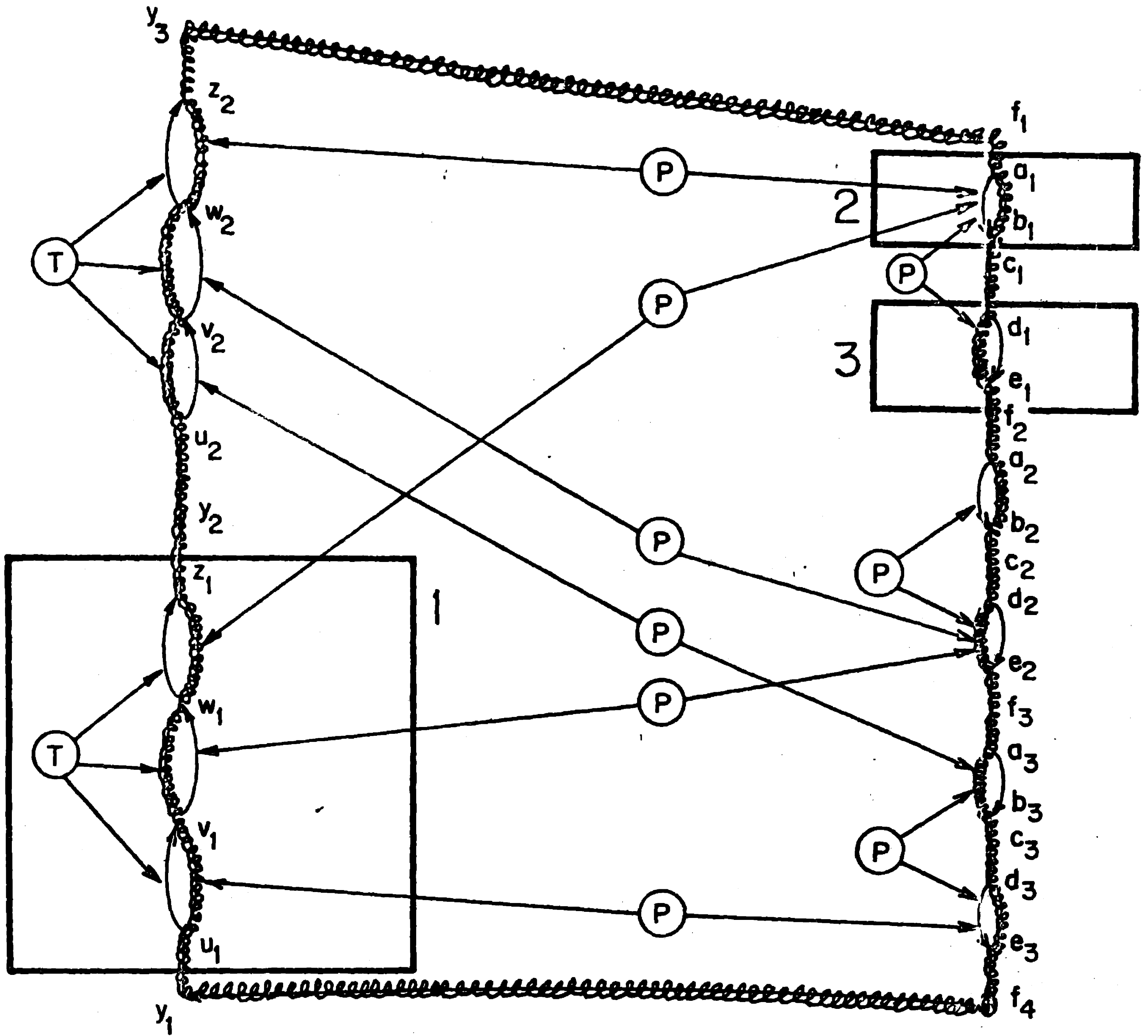
The construction is a rather straight-forward "compilation." For each variable x_j we have five nodes a_j, b_j, c_j, d_j and e_j, two copies of each of the arcs (a_j,b_j) and (d_j,e_j) and one copy of each of the arcs (b_j,c_j) and (c_j,d_j) (see Figure 16): The "left" copies of (a_j,b_j) and (d_j,e_j) 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 16). 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 16.

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 true otherwise if the right copy of (a_i,b_i) and the left of (d_i,e_i) are traversed, x_i is false. 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 16, x_1 = false, x_2 = false, x_3 = true.)

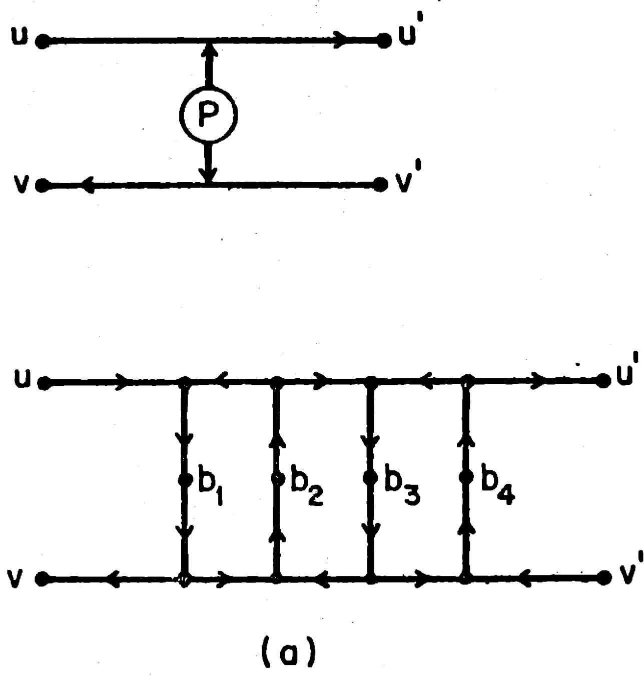
Conversely using any truth assignment that satisfies F, we can construct, as above, a feasible Hamilton circuit for D. This proves the lemma. □

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 exclusive-or constraint) and T (an or 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 17a and b can be used in the proof of Lemma 1 instead of the P and T connectives, respectively [27]. Garey, Johnson and Tarjan applied this technique to planar, cubic, triconnected graphs [16], and another application appears in [26].



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

Figure 16



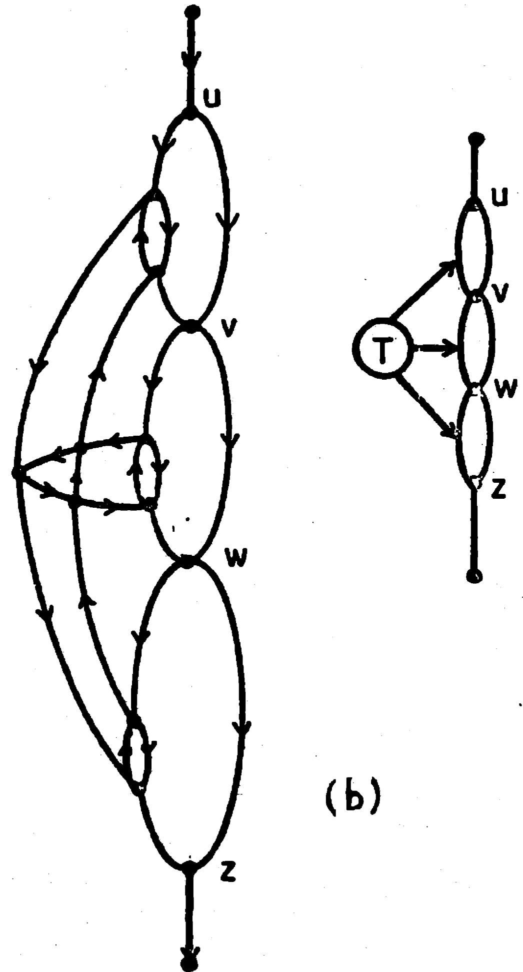


Figure 17

In Figure 17a the Hamilton circuit either passes through u and u' or through v and v'. In Figure 17b no Hamilton circuit can contain all the arcs (u,v), (v,w) and (w,z).

Our proof of Theorem 2.2.2 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 $(\bar{x}_1 \vee x_2)$ $(\bar{x}_2 \vee x_3)\ldots(\bar{x}_k \vee x_1)$ -- which are, of course $x_1 \equiv x_2 \equiv x_3 \ldots \equiv x_k$ in conjuctive 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 orignal 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 17, 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.

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

a) b has indegree and outdegree one.

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

Then b is called a <u>bond</u>. Removal of all bonds from D divides D into several (weakly connected) <u>components</u>. For example b_1, b_2, b_3, b_4 are bonds in Figure 17a, the y nodes, the f nodes and the c nodes are bonds in Figure 16.

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

<u>Proof</u>. Assuming that each component $F_i (i=1,\ldots,k)$ of D can be realized by a job system J_i, we shall show that D itself can be realized by a job system J. For each 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), $(b_j,v) \varepsilon A$. 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 2.2.2. As mentioned earlier, the construction is essentially that pictured in Figure 16 and our P- and T-digraphs are similar -- although not identical -- to the ones shown in Figures 17a and 17b. 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 16 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 bounds as are the b_1, b_2, b_3, b_4 nodes of the P-digraph in Figure 17a.

In Figure 18a we show the component corresponding to each clause of F, as well as its realization by a job system $\mathcal{J}$ shown in Figure 18b. We omit here the straight-forward but tedious verification that, indeed, the component shown is $D(\mathcal{J};\{2,3\})$. This verification can be done using the inequalities in 18c. We 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 19a we show the component that corresponds to an unnegated variable occurring twice in F. Again the quadruples (2,3,4,5), (6,7,8,9), (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 19b, via the appropriate inequalities.

The lower part of 19a shows the component that corresponds to negations of variables and is realizable in a similar manner as in 19b.

The remaining argument is to the effect that copies of these three components, when properly connected via bonds as shown in Figure 16, 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 19a) 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 19a and (5,2) in Figure 18a can never participate in a Hamilton circuit and are therefore irrelevant.

Secondly lines such as (1,6) in Figure 18a do not affect the existence of Hamilton circuits, because a path from 1 to 6 already exists and passes through two other nodes. Thirdly more attention has to be paid to arcs like (13,8) and (13,4) of Figure 19a. Suppose we start with the graph not containing any arcs of the form (13,8) or (13,4). If it has a Hamilton circuit so does the graph with all these edges added on. If it does not have a Hamilton circuit by introducing say (13,8) in a Hamilton

path we are forced to include (15,16) and (1,14) of Figure 19a and exclude (5,6), (1,8), (13,14). This makes it even more difficult than previously to traverse nodes 9,10,11 in the copies of Figure 18a. 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 19a), corresponding to x, or the arc (15,16) (Figure 19a), corresponding to $\bar{x}$, is traversed. The former means that x is <u>false</u>, the latter that is is <u>true</u>, then only clauses having at lest one literal true shall have the corresponding nodes 9, 10, 11 (Figure 18a) traversed. Thus a Hamilton circuit exists in D if and only if F is satisfiable and the sketch of our proof is completed. □

Certain remarks are in order. If we wished to prove the same theorem for (5,0)-FS we would have no extra arcs in the realizations of figures 18 and 19. From the three relations between processing times used to define arcs $\beta_2 = \gamma_1$ isolates the different parts of the graph and $\alpha_2 \leq \beta_1$, $\gamma_2 \geq \delta_1$ create the proper "gadgets" (of Figure 17).

Also some of the fractional values appearing in the table of figure 18b (e.g. $\gamma_{14} = 4.1$) make it impossible for different bonds to interact with each other if we use the realization of Lemma 3 for bonds.

Now we can prove Theorem 2.2.1:

<u>Proof of Theorem 2.2.1</u> We shall reduce the (4;{2,3})-Hamilton circuit problem to it. Let J be a job system constituting an instance of this problem. It is evident from the proof of Theorem 2.2.2 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

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

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 $m \geq 4$. □

A summary of the proof in 2.2 is presented for clarity:

P1 M=4 /F/ 0-BUFFERS

P2 M=4 /F/ 0-BUFFERS/ ARE 2ND & 3RD MACH. SATURATED?

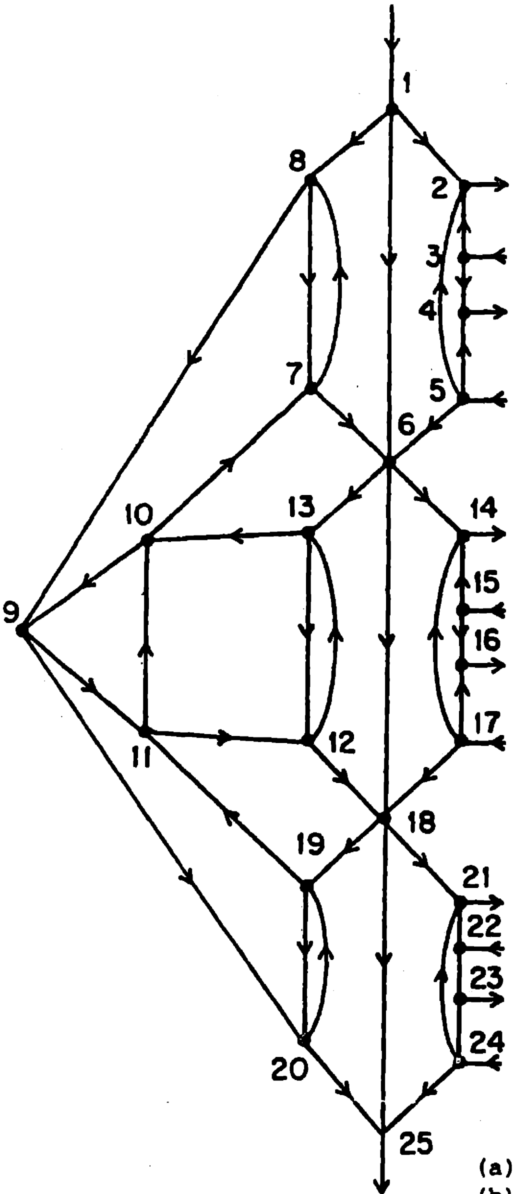
P3 FINDING HAMILTON CIRCUITS IN SPECIAL GRAPHS.

P4 3-CNF SATISFIABILITY PROBLEM IN RESTRICTED FORM

P5 GENERAL 3-CNF SATISFIABILITY (NP-COMPLETE)

$$P5 \propto P4 \propto P3 \propto P2 \propto P1$$

Sketch of proof of NP-completeness of P1



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

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

$\beta_3, \beta_5 \geq \alpha_4$

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

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

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

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

$\delta_1, \delta_7 \leq \gamma_8$

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

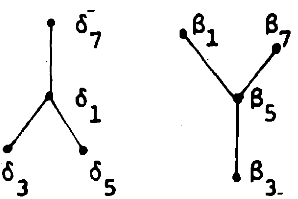
$\delta_{10}, \delta_8 \leq \gamma_7$

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

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

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

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



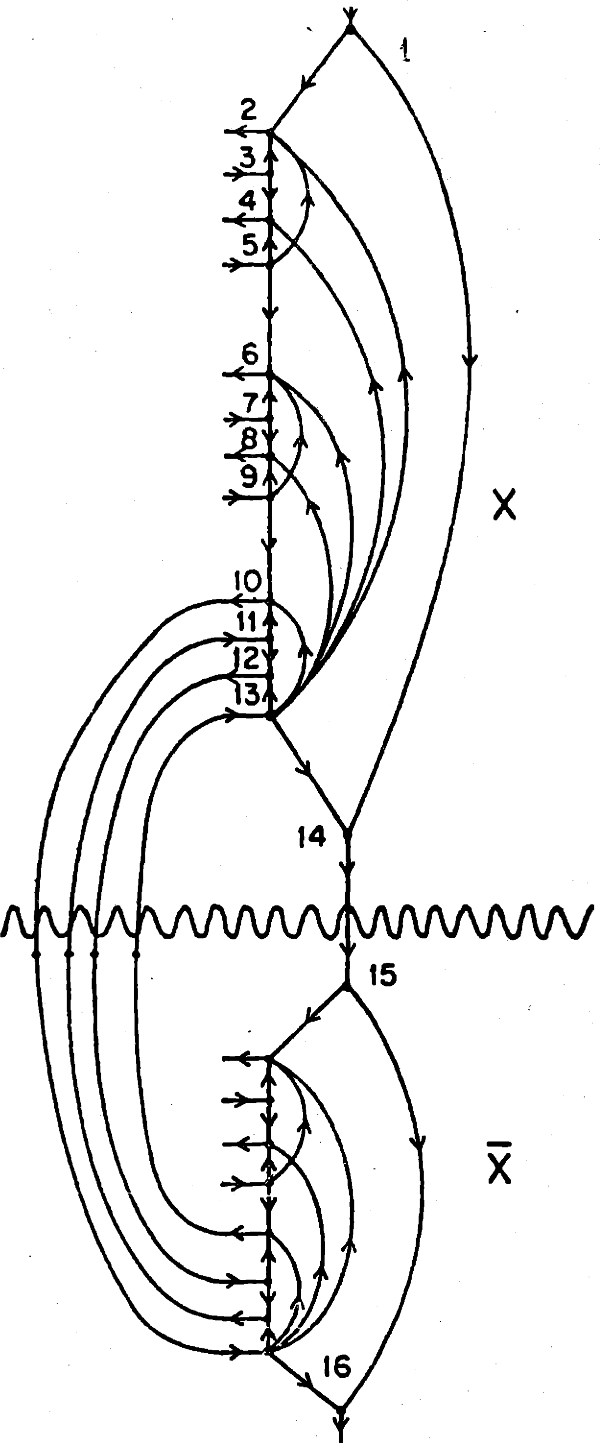
(c)

	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	14.1	0
15	0	5.1	200	5
16	4	200	9.1	0
17	0	10.1	200	5
18	9	200	300	10
19	11	18	300	0
20	0	300	18	15
21	4	300	14.2	0
22	0	5.2	300	5
23	4	300	9.2	0
24	0	10.2	300	5
25	9	300	400	0

(b)

Figure 18

(a) component for clause
(b) table of values (x10 for integers)
(c) inequalities and lattice of β's and δ's for nodes 1 to 10



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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\gamma_1 = \beta_2 = \gamma_3 = \beta_4 = \gamma_5 = \beta_6 = \gamma_7 =$$
$$= \beta_8 = \gamma_9 = \beta_{10} = \gamma_{11} = \beta_{12} = \gamma_{13} = \beta_{14}$$

	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	16	0

(b)

β	δ
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 19

2.3 Three Machine Flowshops

Our analysis up to this moment has left one important 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 [24] - 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. As pointed out in the previous paragraph for 2 jobs in the 4 machine case to determine an arc one equality and two inequalities are needed. The equality acts as a signature isolating "gadgets".

JOB 1 $(\alpha_1, \beta_1, \gamma_1, \delta_1)$

JOB 2 $(\alpha_2, \underbrace{\beta_2, \gamma_2}_{\text{SIGNATURE}}, \delta_2)$

1 ⟶ 2

$\alpha_2 \leq \beta_1$

$\gamma_2 \geq \delta_1$

$\beta_2 = \gamma_1$

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 $\mathcal{D}(m;\{1,2,\ldots m\})$ 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 [34]. Consequently, this class of problems can be solved in linear time. This leaves us with the $|K| = 1$ case.

Determining whether $\mathcal{D}(m;K)$ has a Hamilton circuit where $|K|=1$ is open. This is because for each pair of jobs to determine an arc in the graph we have m-1 inequalities. These inequalities propagate in

the graph through transitivity and make it difficult to isolate its components. For m=3 if $J_1 = (\alpha_1, \beta_1, \gamma_1)$ $J_2 = (\alpha_2, \beta_2, \gamma_2)$ in order to have the arc (1,2) in $D(J;\{2\})$ we must have $\alpha_2 \leq \beta_1$ and $\gamma_1 \leq \beta_2$. The lower limits in order to obtain a meaningful equality are $|K| \geq 2$ and $m \geq 4$.

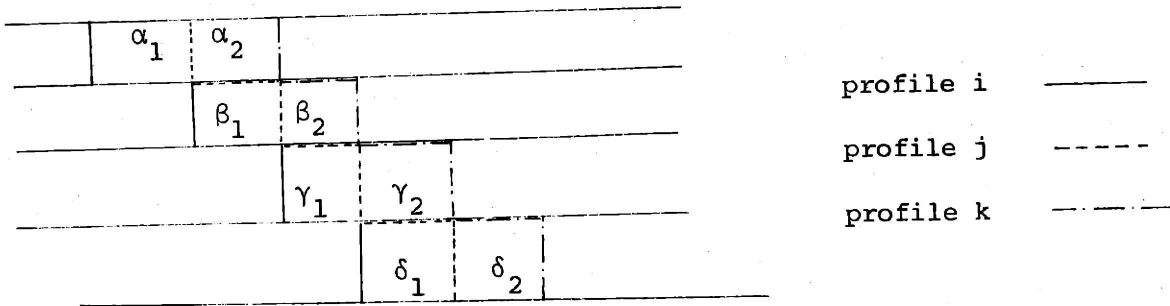
Let us now look at the solvable cases:

Theorem 2.3.1 [34] Determining whether graphs in $\mathcal{D}(m;\{1,2,3,\ldots,m\})$ have Hamilton circuits can be done in $O(n)$ time.

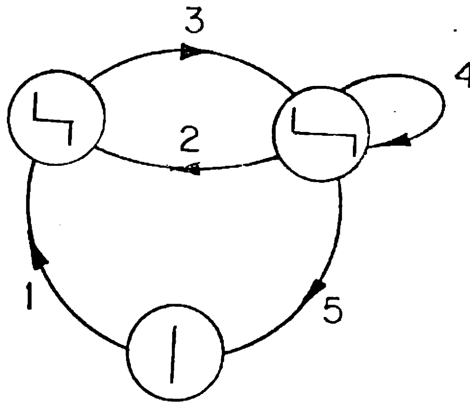
Proof In order to place one job after the other they must fit exactly. Therefore, if each job is viewed as a transition from one "profile" to another as exhibited in Figure 20 we can use the following approach.

1) Construct digraph $D = (V,E)$, where each $v \in V$ corresponds to a profile and each $e \in E$ to a job which takes us from one profile to another.

2) Find an Euler circuit in D (circuit passing through all edges exactly once). This can be done in $O(|E|) = O(n)$



(a) Figure 20



(b) Figure 20

An Example of D = (v,E)

Using similar ideas we have:

Theorem 2.3.2 Determining whether graphs in $\mathcal{D}(m;K)$, where $|K| = m-1$, have Hamilton circuits is solvable in $O(n\log n)$.

Proof: We will illustrate the argument that reduces the $\mathcal{D}(3;\{2,3\})$ problem to the (2,0)-FS problem. It will be obvious that a similar coding technique can be used to reduce an arbitrary $\mathcal{D}(m,K)$, $|K| = m-1$ problem to the (2,0)-FS problem. As will be shown in the next chapter the (2,0)-FS problem is solvable in $O(n\log n)$ time.

Given a job system J, where $J_i = (\alpha_i, \beta_i, \gamma_i)$, we wish to determine whether there exists a schedule which is saturated on the 2nd and 3rd machines. Consider the job system J', where J'_i corresponds to J_i, and $J'_i = (M\beta_i + \alpha_i, M\gamma_i + \beta_i)$, where $M = \sum_i \beta_i = \sum_i \gamma_i$. We will prove that J' has a schedule on 2 machines of length $M(M+1)$ iff J can saturate the 2nd and 3rd machines.

"$\Rightarrow$" This direction is obvious since $M\beta_j + \alpha_j \leq M\gamma_i + \beta_i$ if $\gamma_i = \beta_j$ and $\alpha_j \leq \beta_i$ (These are the conditions that J_j follows J_i in a schedule that saturates the 2nd and 3rd machines). Thus by J'_j following J'_i we achieve the $M(M+1)$ length.

"$\Leftarrow$" Since $\mathcal{J}'$ has a schedule of length $M(M+1) = M\sum_i \gamma_i + \sum_i \beta_i$ there exists a permutation r such that $M\beta_j + \alpha_j \leq M\gamma_i + \beta_i$ if J'_j follows J'_i. We will prove that $\gamma_i = \beta_j$ and $\alpha_j \leq \beta_i$. If $\gamma_i \neq \beta_j$ then either $\gamma_i > \beta_j$ or $\beta_j > \gamma_i$. In the first case let I be the idle time on the first machine between J'_i and J'_j. $I = M(\gamma_i - \beta_j) + \beta_i - \alpha_j \geq M + \beta_i - \alpha_j > M - \sum_i \alpha_i =$ (total idle time on the first machine) $= M(M+1) - MM - \sum_i \alpha_i$. This is not possible. In the second case we also reach a contradiction because $M(\beta_j - \gamma_i) \geq M > \beta_i - \alpha_j$. Thus $\gamma_i = \beta_j$ and $\alpha_j \leq \beta_i$, which demonstrates that $\mathcal{J}$ has a schedule saturating the 2nd and 3rd machines. $\square$

Let us now look at a particular case we have ignored. "Three machine flowshops with one 0 buffer and one ∞ buffer". The technique 2.1 can be applied to show that minimizing makespan is NP-complete for flowshop systems, such as 3-machine flowshops with 0 buffer between machines 1 and 2, and ∞ buffer between machines 2 and 3.

Theorem 2.3.3. The $3,\begin{pmatrix} b_1=0 \\ b_2=\infty \end{pmatrix}$-FS problem is NP-complete .

Proof: 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 ajob $B_i = (1, b_i, 0)$ 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 $\mu(S) < L$ iff there is no idle time on the second and third machines, yet there can be idle time on the first.

1	//////	1	//////	K_2 m	1	...
K_1 1	b_i		b_j	$a_i + m$	K_2 1	...
K_0 1	K_1 c + m + 1					...

Figure 21

The K_i jobs create slots of length c+m on the second machine's Ghannt chart. If the partition problem has a solution these slots are filled in the schedule as is obvious from Figure 21. If conversely we have a schedule the second machine must be saturated. As in Figure 21 only a task with length a_i+m can be present. The restrictions $1 < c/4 < b_j < c/2 << m$ make it necessary to use two B_i tasks before each A_i. □

If we are concerned with no idle time schedules on the first two machines it is not obvious whether the question can be answered efficiently. Maybe we can code the two first machine times into one

and use Johnson's algorithm [20] for $(2,\infty)$-FS. Perhaps the problem is intractable. (The coding of Theorem 2.3.2 does not work in a straightforward fashion).

So we have determined certain fundamental differences between a machine and a buffer, which show up even if we compare the (2,1)-FS and the (3,0)-FS problem (to go from one to the other we replace a buffer by a machine).

1) A machine cannot replace a buffer by allowing jobs to wait on it after they terminate (Figure 4).

2) A buffer needs the FIFO nonbypassing restriction to behave like a machine with respect to permutation schedules.

3) On (2,1)-FS saturated schedule questions are NP-complete, whereas for (3,0)-FS saturated schedule questions are easy or open.

Let us briefly comment on possible uses of the flow time criterion in the "no-wait" problems examined. The (2,0)-FS problem under a flow time criterion is open [24]. We conjecture that although it will be algebraically tedious our methodology in 2.2 can prove NP-completeness for the (3,0)-FS with the flow time criterion.

The interesting open problem of this section is "Determining the existence of Hamilton circuits in $\mathcal{D}(3;\{2\})$ graphs". (See Figure 22).

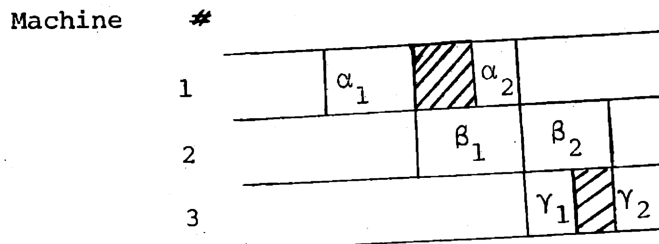


Figure 22 - J_1 can precede J_2

3. AN APPROXIMATION ALGORITHM FOR TWO MACHINE FINITE BUFFER FLOWSHOPS

3.1 The (2,0)-FS Problem and the Gilmore-Gomory Algorithm

As was previously indicated the (2,0)-FS problem is solvable by use of a fairly complicated algorithm [10]. In this section we will re-examine this procedure. We will attempt to explain its operation, prove that the (2,0)-FS problem is $\Omega(n\log n)$ lower bounded and that the timing of the algorithm is $O(n\log n)$ and not $O(n^2)$ as was implemented in the literature.

Theorem 3.1.1. The (2,0)-FS problem is $\Omega(n\log n)$.

Proof. We will prove this by reducing sorting to the (2,0)-FS problem Since we do not know how to do sorting in less than $O(n\log n)$ this is a convincing lower bound. Given a list of n numbers $(x_1,\ldots,x_n)$ to sort consider the following instance of the (2,0)-FS problem.

Is there a schedule for the set of jobs $J_i = (x_i,x_i)$ $1 \le i \le n$, $J_0 = (0,x_{min})$, $J_{n+1} = (x_{max},0)$ where x_{min} and x_{max} are the minimum and maximum elements respectively from our list and the required schedule length is $L = \sum_{i=1}^{n} x_i + x_{max}$. If the list is sorted we use (π = identity permutation) for our schedule. If we have a schedule it must be saturated on the first machine. Therefore if J_j follows J_i we have have $\beta_i = x_i \le x_j \le \alpha_j$. Thus we have sorted the list.

Let us describe the Gilmore-Gomory Algorithm as applied to the (2,0)-FS scheduling problem. Given a list of n pairs (α_i,β_i), of which

one pair is the starting job (0,0) we go through the following steps.

Algorithm G-G

STEP 1 Sort the β_i's in order of size and renumber the jobs so that with the new numbering $\beta_i \le \beta_{i+1}$, $1 \le i \le n-1$.

STEP 2 Sort the α_i's in order of size.

STEP 3 Determine the permutation $J(P)$ for all $1 \le p \le n$. The permutation J is defined by $J(P) = q$. q being such that α_q is the pth smallest of the α_i.

STEP 4 Compute the numbers $c_J(\alpha_{ii+1})$, $1 \le i \le n-1$ where $c_J(\alpha_{ii+1}) = \max\left\{0, \min\left(\beta_{i+1}, \alpha_{J(i+1)}\right) - \max\left(\beta_i, \alpha_{J(i)}\right)\right\}$

STEP 5 Form an undirected graph with n nodes and undirected arcs connecting the ith and $J(i)^{th}$ nodes $1 \le i \le n$.

STEP 6 If the current graph has only one component go to STEP 8. Otherwise select the smallest value $c_J(\alpha_{ii+1})$ such that i is inone component and i+1 in another. Break ties arbitrarily.

STEP 7 Adjoin the undirected arc $\{i, i+1\}$ selected to the graph, creating fewer components, go to STEP 6.

STEP 8 Divide the edges added in STEP 7 in two groups. Those for which $\alpha_{J(i)} \ge \beta_i$ go in group 1, those for which $\beta_i > \alpha_{J(i)}$ go in group 2.

STEP 9 Sort the edges in group 1 so that i_1 is the largest index, i_2 the second largest etc. (i_1 is the index of $\{i_1, i_1+1\}$).

STEP 10 Sort the edges in group 2 so that j_1 is the smallest index, j_2 the second smallest etc. (j_1 is the index of $\{j_1, j_1+1\}$).

STEP 11 The minimal tour is obtained by following the ith job by $J^*(i)$

$$J^*(i) = J\pi_{i_1, i_1+1}\pi_{i_2, i_2+1}\cdots\pi_{i_\ell, i_\ell+1}\pi_{j_1, j_1+1}\pi_{j_2, j_2+1}\cdots\pi_{j_m, j_m+1}(i)$$

where π_{pq} is defined as $\begin{cases} \pi_{pq}(P) = q \\ \pi_{pq}(q) = P \\ \pi_{pq}(i) = i, \; i \neq p,q \end{cases}$

and the order of applying the permutations is from right to left in the sequence given.

The Intuition behind the algorithm is the following :

In steps 1-3 the optimal permutation J is found, which solves the assignment problem. Instead of finding an optimal Hamilton tour, what is determined is the optimal set of cycles that covers the graph on which we wish to find the ATSP solution. (This is also called Gilmore-Gamory bipartite matching).

The purpose of the rest of the algorithm is to carry out a series of interchanges, which convert the permutation J into a tour J^*. The interchanges to be executed will be chosen by finding a minimal spanning tree and must be carried out in a special order if the resulting tour J^* is to be of minimal cost.

The $c_J(\alpha_{ii+1})$ represent the costs of interchanges π_{ii+1} for the permutation J. The proof in [10] quarantees first that interchanges of this form can construct a spanning tree (steps 6-7) for the graph of step 5 (where each cycle covering the graph corresponds to a node) therefore these interchanges can produce a Hamilton tour. Second if

interchanges are made in a proper order (steps 8-10) then the resulting tour is the optimal tour and each interchange actually contributes $c_J(\alpha_{ii+1})$ to the cost of the tour.

The Timing of the Algorithm is as follows:

- The sorting steps 1-3 are 0(nlogn)
- Computing n costs in step 4 is 0(n)
- Steps 5-7 can actually be implemented using a UNION-FIND [ ] approach. Let us sort the $c_J(\alpha_{ii+1})$. We construct the graph of step 5 and the different components form the different sets. We now scan the possible interchanges in the sorted list and FIND, which sets the i and i+1 of the scanned $c_J(\alpha_{ii+1})$ belong to. If they belong to different sets we do a UNION. Obviously, what is costly here is only the 0(nlogn) sorting.
- Steps 8-10 consist of sorting 0(nlogn)
- Finally determining the optimal tour from step 11 can be done by interchanging 0(n) pointers on an array containing 1 to n.

Therefore Algorithm GG is 0(nlogn).

The Implementation of Algorithm CG:

A simple LISP program has been wirtten for the GG Algorithm and is contained in the Appendix. Let us briefly comment on the different functions. The input is a list of pairs $((\alpha_1\beta_1)\ (\alpha_2\beta_2)\ldots)$

LESSPA-P1A-P1B-P1- P2-P3 compute steps 1-3

P4	computes step 4
S1	computes step 5
S23T1	computes step 6-8
T23	computes steps 9-10
T4	computes step 11

GIGO actually combines the above and the optimal list is the output of:

(GIGO 'L)

Also an array DISTANCE contains the TSP distances (the numbering of elements in the array corresponds to the Gilmore-Gomory sequence).

3.2 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 to obtain a permutation π of $\mathcal{J}$.

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

Let $F(i,j)$ be the finishing time of the ith job on the jth machine. If for simplicity of notation our schedule is based on (π = identity permutation) we have the following equations*

$$F(0,1) = 0 \qquad 1 \leq i \leq n$$

$$F(i,1) = \begin{cases} \sum_{j=1}^{i} \alpha_j & i < b+2 \\ \max(F(i-b-1,\, 2),\, F(i-1,\, 1) + \alpha_i) & i \geq b+2 \end{cases}$$

$$F(i,2) = \max(F(i-1,\, 2),\ F(i,1)) + \beta_i$$

The makespan is $F(n,2)$.

---

*These are used slightly modified in [9] where the first machine can act as temporary storage.

The approach of [9] for the solution of these equations is based on dynamic programming. The two basic difficulties are large storage requirements for large numbers of jobs (after 15 jobs one is forced to use heuristics) and in the case of large job sets reduced accuracy for small b. Our approach has no storage or time problems and can handle very large batches easily. The average accuracy for any number of jobs is less than 5% for b=1. Also the approach indicates how larger groups of machines can be handled. Flowshops as in Figure 1 with intermediate storage can be treated as 0 buffer flowshops formulated as ATSP's and solved (Chapter 4 proposes some heuristics) then the permutation determined can be used for the small finite buffer case. Similar ideas can be applied to job shops as in Figure 3 since bounds based on ATSP's are used for the branch and bound approach of [24].

Analysis of Algorithm A

Let $\mu_b(J)$ be the shortest possible makespan of a job system J using b buffers , as will be demonstrated in the next paragraph:

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

In other words, the use of b buffers can save up to b/2b+1 of the time needed to execute any job system.

If $\mu_A(J)$ is the makespan produced by Algorithm A it follows that

$\frac{\mu_A(J)}{\mu_1(J)} \leq \frac{2b+1}{b+1}$. However it does not follow directly that the $\frac{2b+1}{b+1}$ ratio is achievable. Figures 23-24 show how these bounds can be achieved for b=1. The same heuristic could be used for the (2,b)-FS problem and a

similar worst case example Figure 25, 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.

The worst-case job system for algorithm A with b=1 is shown in Figure 23. In Figure 23a we show the optimum 1-buffer schedule with $\mu_1(J) = 2 + \varepsilon + \delta$. It can be checked that the application of A yields the schedule in Figure 23b, with $\mu_A(J) = 3 + \delta - \varepsilon$. When $\varepsilon < \delta \to 0$ we have an asymptotic ratio of 3/2.

The equivalent $ATSP_2$ is shown in Figure 24.

What we can notice about Figure 24 is that some second to optimal tours could possibly give us by "squeezing" out the idle time the optimal 1 buffer schedule (e.g., the schedule $J_1 - J_3 - J_4 - J_2$ has length $3+\delta$ yet for 1 buffer it is very close to optimal). Maybe by checking the permutation algorithm A produces and looking at finite fixed length sequences of jobs (say 4 jobs) and manipulating these jobs locally we might be able to guarantee a $\frac{5}{4}$ instead of a $\frac{3}{2}$ performance.

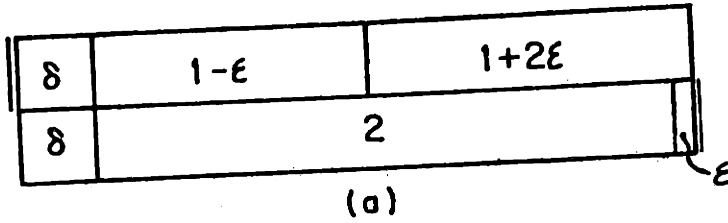
In Figure 25 we give the worst case example for the (2,b)-FS case. Figure 25a has the $ATSP_2$ formulation where nodes J_2 to J_{1+b} have identical characteristics. Figure 25b contains the optimal schedule. We must have

$$\frac{1}{b+1} \gg \delta > \varepsilon \ , \quad \frac{1}{b+1} \gg 2\varepsilon \quad \text{and} \quad 4b\varepsilon > \delta$$

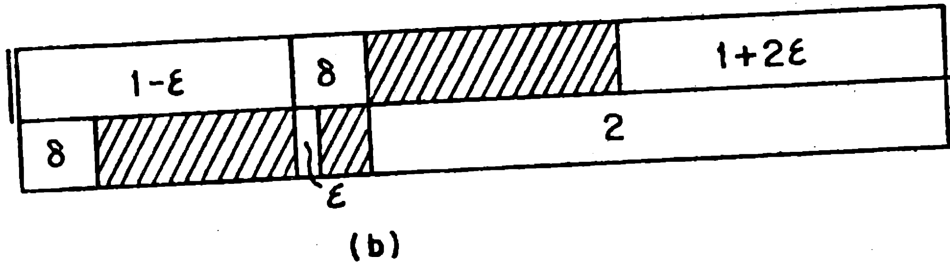
The optimal 0 buffer schedule is $(J_1 - J_2 - J_3 \ldots J_{b+3})$ and cannot be compressed. The idle time is

$$\frac{1}{b+1} - \varepsilon - \delta + (b-1)\left(\frac{1}{b+1} - 2\varepsilon\right) + \delta - \varepsilon = b\left(\frac{1}{b+1} - 2\varepsilon\right)$$

$1 \gg 4\varepsilon > \delta > \varepsilon$



(a)



(b)

Figure 23

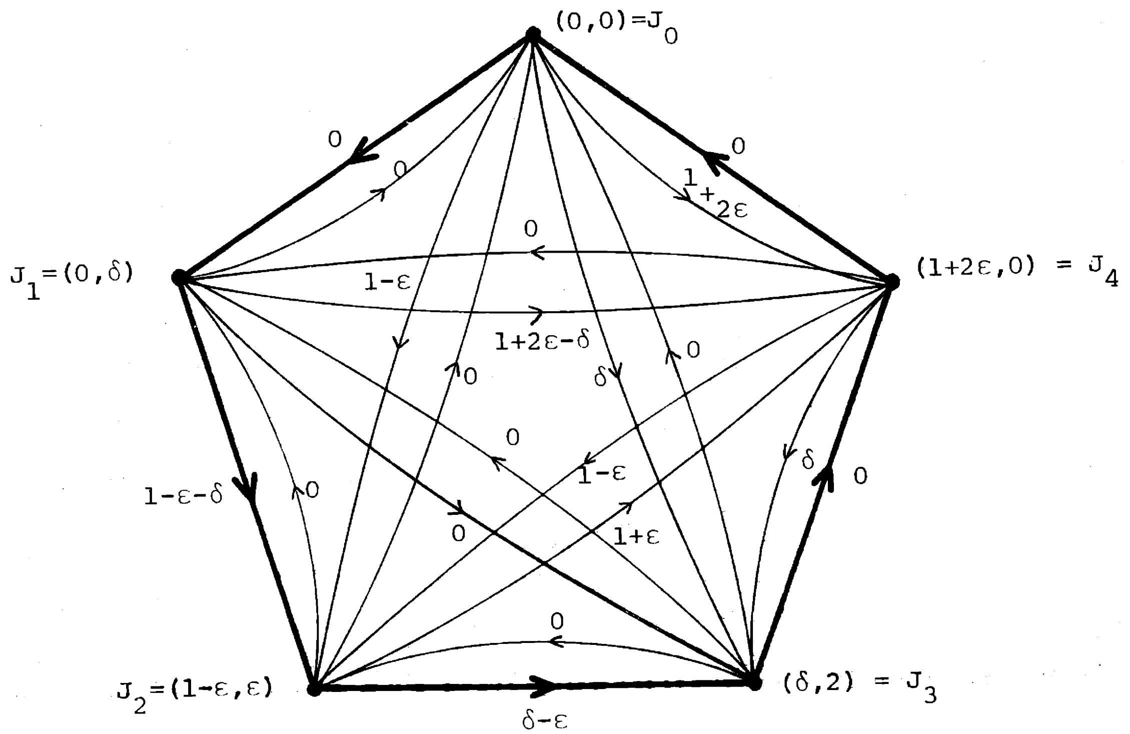
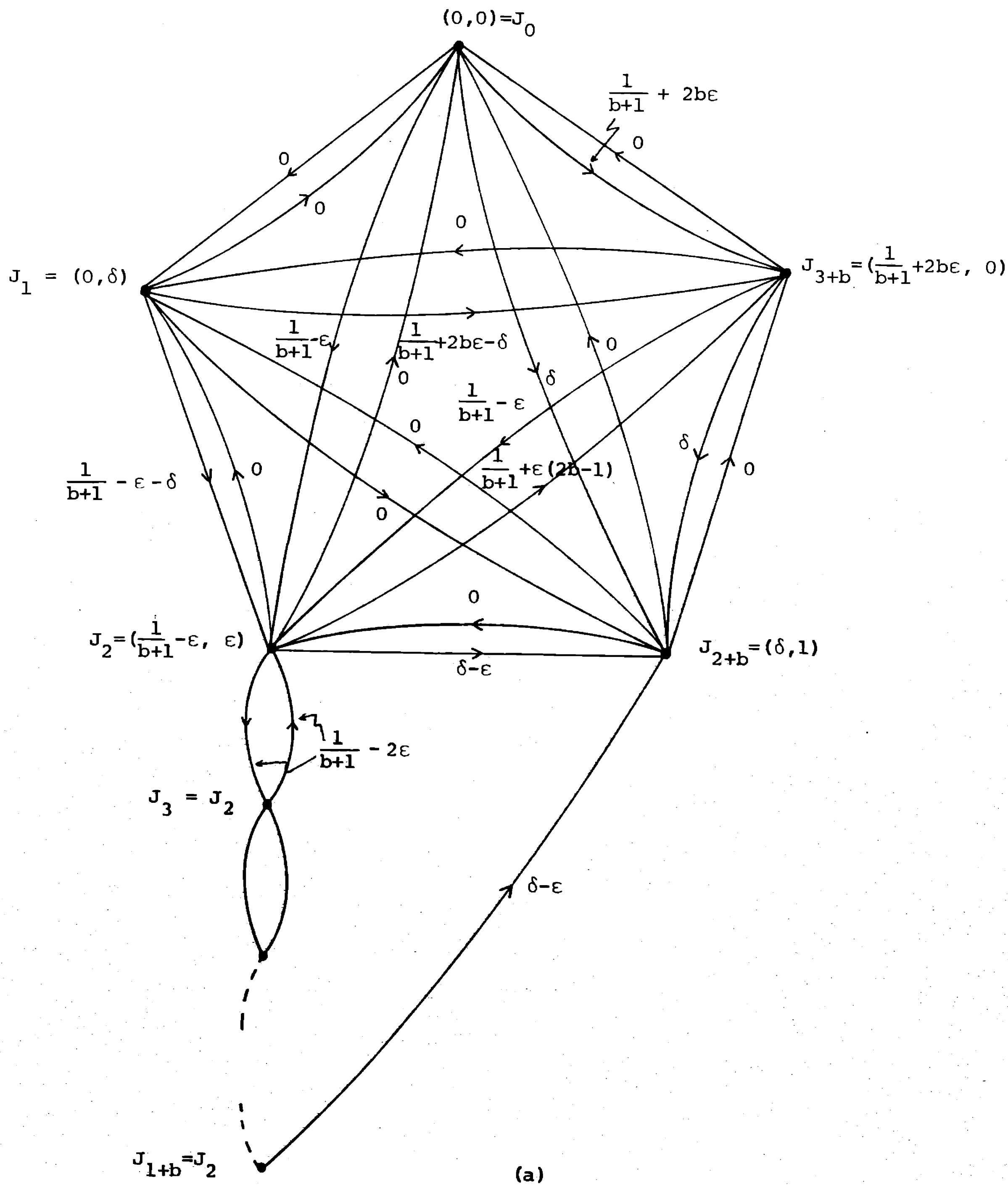


Figure 24



δ	$\frac{1}{b+1}$ - ε		. . . .	$\frac{1}{b+1}$ + 2bε			
δ	1				ε		...

b (over the $\frac{1}{b+1}$ - ε ... cells); b (under the ε ... cells)

(b)

Figure 25

Lemma: For any job set J we can find a job set J' with the following properties.

1) $\mu_1(J) = \mu_1(J')$.

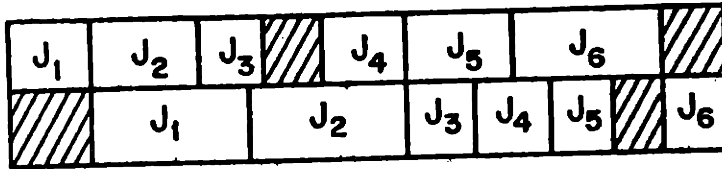
2) The optimal schedule of J' is saturated.

3) Algorithm A performs worse on J' than on J.

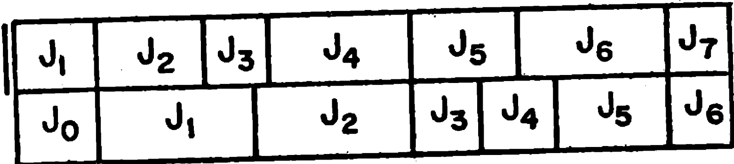
Proof: Given J and its optimal schedule S all we have to do is fill in the idle times as in Figures 26(a) and (b). Since for two machines an equivalent formulation is with distances $P_{ij} = \max(\alpha_j, \beta_i)$ (see example 1.2.3) it is obvious that algorithm A performs worse than before. What we should note is that we cannot always saturate the machines as easily.

For example, in the 3 machine case with $b_1=1$, $b_2=0$ the above argument does not hold. We cannot saturate the system in Figure 26(c) without making some of the lengths of jobs smaller. As a result for a rigorous analysis we might have to look at more than just the saturated schedules.

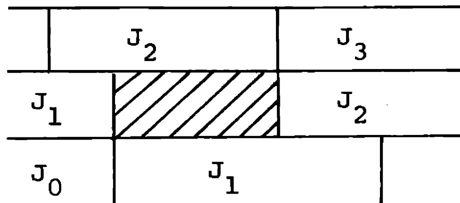
Since we have to look only at saturated schedules for the 2 machine case it is:1) easier to analyze upper bounds (see section 3.3). 2) It is possible to generate saturated schedules randomly and simulate. Then the performance of algorithm A is tested on the harder cases and we know the optimal a priori. 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 Appendix contains a listing of programs used for this simulation.



(a)



(b)



(c)

Figure 26

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	7.8	7

3.3 Bounds for Flowshop Scheduling

Very few bounds exist for approximation algorithms for flowshops [3], [17], [31].

Let L_{opt} be the optimal length schedule for a job system J on a flowshop of m machines. Let the permutations determining a schedule be chosen randomly and let L_R be the length of this random schedule.

<u>Lemma</u>: $\frac{L_R}{L_{opt}} \leq m$

<u>Proof</u>: Let $T_{i,i_2 \ldots i_K}$ stand for the time in a schedule where exactly machines $i_1, i_2, \ldots, i_K$ are occupied. (T represents the L_R and T* the

L_{opt} times). Obviously for each i

$$T_i + \sum_{j \neq i} T_{ij} + \sum_{\substack{j \neq i \\ k \neq i \\ k \neq j}} T_{ijk} + \ldots + T_{12\ldots m} =$$

$$= T_i^* + \sum_{j \neq i} T_{ij}^* + \ldots + T_{12\ldots m}^*$$

Therefore by adding these relations togehter we have:

$$\frac{L_R - L_{opt}}{L_{opt}} = \frac{\sum\limits_{i=j} T_{ij}^* - \sum\limits_{i \neq j} T_{ij} + 2\left(\sum\limits_{i \neq j \neq K} T_{ijk}^* - \sum\limits_{i \neq j \neq K}\right) T_{ijk} + \ldots + (m-1)\left(T_{12\ldots m}^* - T_{12\ldots m}\right)}{T_1^* + T_2^* + \ldots + T_m^* + \sum\limits_{i=j} T_{ij}^* + \ldots + T_{123\ldots m}^*}$$

$$\Longrightarrow \frac{L_R - L_{opt}}{L_{opt}} \leq m-1$$

Even though a lot of heuristics are used in branch and bound methods the only worst case bound for classical flowshops is in [17]. Gonzales and Sahni give an O(mn logn) algorithm H based on Johnson's algorithm (they consider pairs of consecutive machines) with $\frac{L_H}{L_{opt}} \leq \lceil \frac{m}{2} \rceil$. In [3] the case of having two stages, but with multiple processors at each stage is examined (m_1 processors at stage 1 and m_2 at stage 2). The jobs are ordered according to the following rule:

$$J_i \text{ preceeds } J_j \text{ iff } \min\left(\frac{\alpha_i}{m_1}, \frac{\beta_j}{m_2}\right) < \min\left(\frac{\alpha_j}{m_1}, \frac{\beta_i}{m_2}\right)$$

(for $m_1 = m_2 = 1$ this is Johnson's rule)

For this heuristic $\frac{L_H}{L_{opt}} \leq 2 - \frac{1}{\max(m_1, m_2)}$, whereas for a randomly constructed schedule we would have had

$$\frac{L_R}{L_{opt}} \leq 3 - \frac{1}{\max(m_1, m_2)} \quad .$$

The above Lemma holds also for flowshops with buffers, what is significant is the relationship on 2 machines between $\mu_b(J)$ optimal schedule using 6 buffers and $\mu_0(J)$ optimal schedule using 0 buffers [31].

<u>Theorem 3.3.1*</u> $\sup_J \frac{\mu_0(J)}{\mu_b(J)} = \frac{2b+1}{b+1}$

Let us sketch the proof for b=1. For b>1 the proof has a similar flavor but is more complicated [31]. Given the optimal schedule S we may assume that is is saturated without loss of generality (see Lemma of previous pargraph). First, notice that 3/2 is asymptotically achieved from below, just by considering the system $\{(1,\varepsilon), (1,\varepsilon), (\varepsilon,2)\}$, with $\varepsilon \to 0$.

We now consider the schedule S' that uses no buffer, and has the same permutation of jobs as S. In other words, S' is derived from S by "sliding" certain tasks so as to "given up the use of buffer" (Figure 27a). Call a set of consecutive jobs with no idle time in processor 2 between them a <u>run</u>--jobs 1 through k+1 in Figure 27a form a run. To prove the 3/2 bound, it suffices to prove that each run can be appropriately modified so that the total amount of idle time in processor

---

*By Prof. C.H. Papadimitriou

1 attributable to the run R (initially s in Figure 27a) is at most 1/2 $\beta(R)$ where $\beta(R)$ is the total length of the run. If $s \leq \frac{1}{2}\beta(R)$, we are already done. Otherwise, we have

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

and hence

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

Because of the b=1 constraint, it is easy to see that

$$\sum_{i=2}^{k+2} \alpha_i \geq \sum_{j=1}^{k} \beta_k \ . \qquad (2)$$

We conclude that (3) $\alpha_{k+2} \geq \frac{1}{2}\beta(R)$.

We change S' by putting J_{k+1} to the end. The corresponding schedule is shown in Figure 27b. The total idle time in machine 1 that is attributable to the run is now

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

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 this 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

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

thus completing the proof of the bound for b=1.

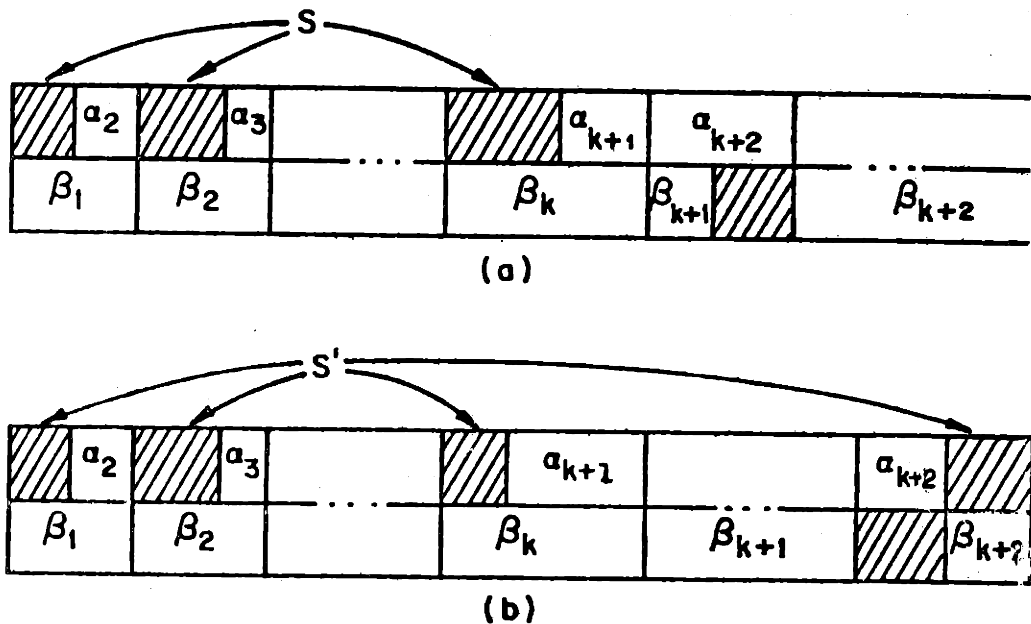


Figure 27

In order to prove the bound for b>1 buffers again the idea is to start from a saturated schedule, "slide" the jobs in order to find the schedule with the same permutation and 0 buffers and determine the runs. Again if the idle time exceeds the desired amount we modify the runs by

matching their jobs together with other runs or by deleting jobs entirely from runs. By a complicated accounting scheme it is possible to prove the $\frac{2b+1}{b+1}$ bound. The details of the proof are contained in [31].

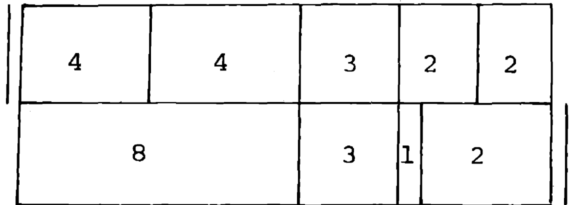
3.4 The Significance of Permutation Scheduling

In section 1.2.4 we have already pointed out the significance of having FIFO buffers that cannot by bypassed by jobs. We should note that the $\frac{2b+1}{b+1}$ bound of section 3.3 is an upper bound on the ratio of the heuristic to the optimal permutation schedule. Let $\mu_b(J)$ be the optimal permutation schedule length using b buffers and $\bar{\mu}_b(J)$ be the optimal schedule length using b buffers (without the restriction of permutation scheduling) we would like to examine $\frac{\mu_b(J)}{\bar{\mu}_b(J)}$. Work has been done for the b=1 case. We conjecture that $\frac{\mu_1(J)}{\bar{\mu}_1(J)} \leq \frac{5}{4}$ and will give evidence in support of this conjecture.

Actually it was impossible to find any example for which $\frac{\mu_1(J)}{\bar{\mu}_1(J)} > \frac{16}{15}$. We should note that picking a job set J it is difficult to determine $\mu_1(J)$ and it seems quite as difficult to figure out if $\bar{\mu}_1(J) \neq \mu_1(J)$. Figure 28 contains the worst example found. (In any optimal saturated schedule (0,8) is bound to come first and the two (4,3)'s to follow in order not to break the buffer constraint. Then (3,1) has to follow for a similar reason. If we had the same permutation of jobs on the second machine we would run into trouble with jobs (2,0) and (2,0). By giving away one unit of idle time we get a permutation schedule of 16 units).

For b>1 we might have more complicated cases such as Figure 29 and it is not obvious that a result for b=1 can be extended to b>1.

Job #	α_i	β_i
1	0	8
2	4	3
3	4	3
4	3	1
5	2	0
6	2	0



Optimal Nonpermutation Schedule $\mu_1(J) \neq \overline{\mu}_1(J)$

Figure 28

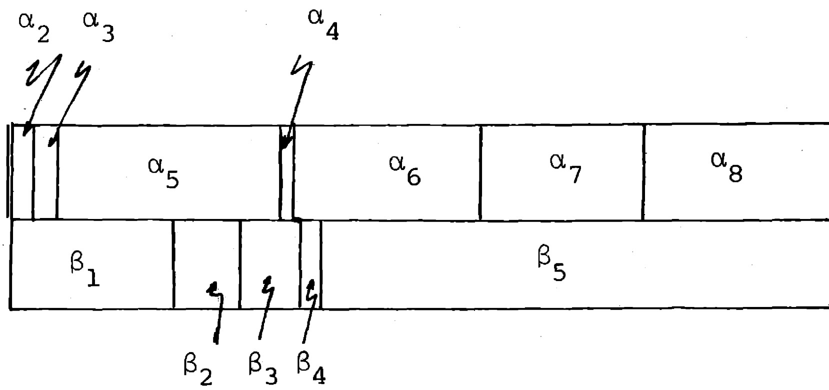


Figure 29

i	α_i	β_i
1	0	5
2	1	3
3	1	3
4	.5	1
5	8	44
6	15.5	0
7	15	0
8	15	0

Remark: Let us note that for a bound on $\frac{\mu_1(J)}{\overline{\mu}_1(J)}$ we need only examine cases where $\overline{\mu}_1(J)$ is saturated. Otherwise, we fill it in properly and obviously the new $\mu_1(J')$ will be worse than the old one.

In Figure 30 we have the different patterns that can possibly occur for b=1 and saturated schedules. In Figure 30a J_2 preceeds J_3 on machine 1, but we have the reverse order on machine 2. Yet job J_2 can remain in the buffer and be bypassed by many jobs as shown in Figure 30b. We call the shaded region in Figure 30a a "switch". The length of the switch is $\alpha_2+\alpha_3+\beta_3+\beta_2$. Switches might correspond to entirely different parts of the schedule or might be close together as exhibited in Figures 30C and 30d, in this case we say that "switches interact".

Lemma: If the optimal schedule contains only switches which do not interact then $\frac{\mu_1(J)}{\overline{\mu}_1(J)} \leq \frac{5}{4}$.

Proof: We can assume that the optimal schedule is saturated, also that $\mu_1(J) \neq \overline{\mu}_1(J)$, that is every modification of the optimal schedule that leads to a permutation schedule introduces idle time.

All the switches do not interact; suppose that they have the form in Figure 30a. Using the notation on this figure we can modify the schedule locally at each switch to produce a permutation schedule in three ways:

(a) Schedule the sequence of jobs $-J_1-J_2-J_3-$, that is, invert β_2 and β_3 on machine 2.

(b) Schedule the sequence of jobs $-J_1-J_3-J_2-$, that is, invert α_2 and α_3 on machine 1

(c) Schedule the sequences of jobs $-J_1-J_2-\ldots J_3$, that is put (α_3,β_3) at the end of the schedule.

Since the modifications have to introduce idle time

$$\begin{aligned} \alpha_2 > \beta_1 \geq \alpha_3 \\ \beta_2 > \alpha_4 \geq \beta_3 \end{aligned} \qquad (*)$$

The idle times introduced in the three cases are:

(a) $\beta_2-\alpha_4 \leq \beta_2-\beta_3$

(b) $\alpha_2-\beta_1 \leq \alpha_2-\alpha_3$

(c) $\max(\alpha_3,\beta_3)$

So actually we wish to determine K s.t. (*) hold and

$$I = \min(\alpha_2-\alpha_3,\beta_2-\beta_3, \max(\alpha_3,\beta_3)) \leq K(\alpha_2+\alpha_3+\beta_2+\beta_3)$$

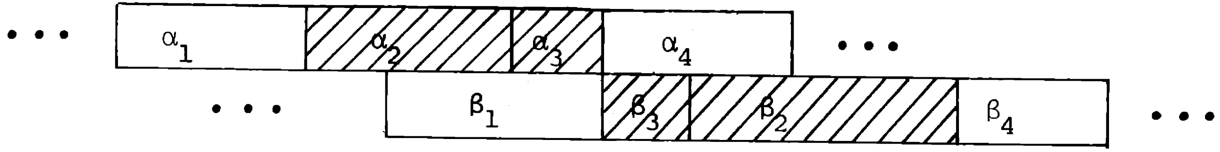
By exhaustively examining all cases we have $K=\frac{1}{4}$. This can be proven also in a simple manner:

Let $\alpha_2-\alpha_3 = x_1 \quad \beta_2-\beta_3=x_2 \quad \max(\alpha_3,\beta_3) \leq \alpha_3+\beta_3 = x_3$

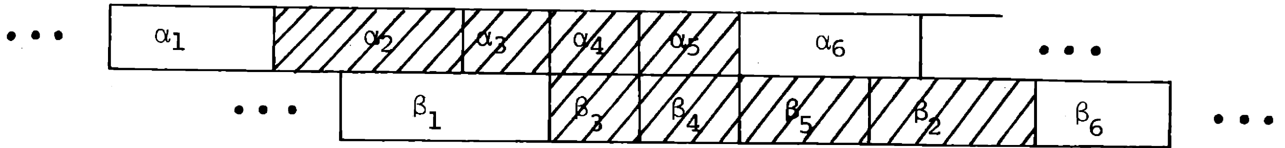
$$\alpha_2+\alpha_3+\beta_2+\beta_3 = x_1+x_2+2x_3$$

$$I \leq \min(x_1,x_2,x_3) \leq \frac{1}{4}(x_1+x_2+2x_3)$$

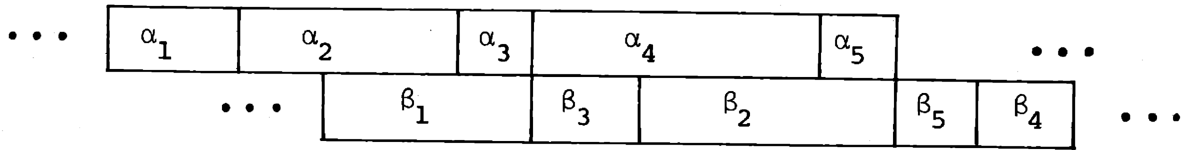
Also note that $I \leq \min(x_1,x_2,x_3) \leq \frac{1}{2}(x_1+x_2) \leq \frac{1}{2}(\alpha_2+\alpha_3+\beta_3)$. Thus the idle time introduced by the switch is at most $\frac{1}{4}$ of its length. This almost proves the lemma since the switches do not interact. Figure 31 shows that the bound is attainable.



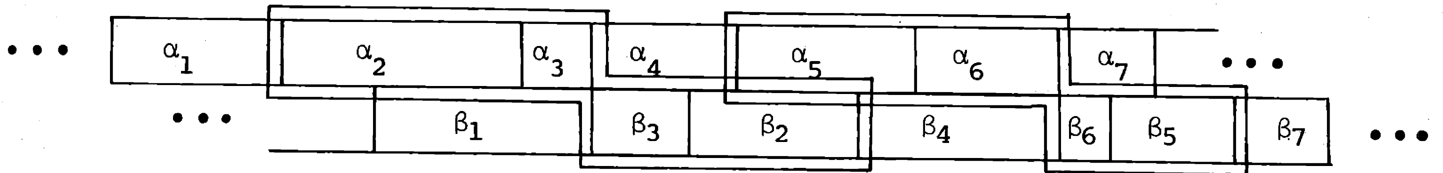
(a)



(b)



(c)



(d)

Figure 30

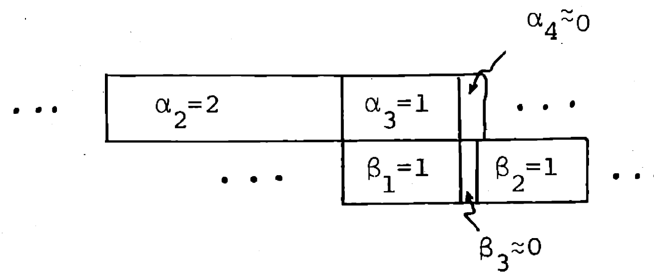


Figure 31

The only other case that must be examined is switches of the form of Figure 30b. Change (a) introduces idle time which at worst is $\beta_2+\beta_3+\beta_4-\alpha_4-\alpha_5-\alpha_6 = \beta_2-\alpha_6$, therefore the analysis is similar as before.

The interesting question that arises is what happens if switches interact, then our accounting scheme of the lemma does not work. Suppose that we pictorially represent switches as in Figure 32. (The x_1, x_2 and y_i's denote noninteracting lengths and the z_i's interacting lenths).

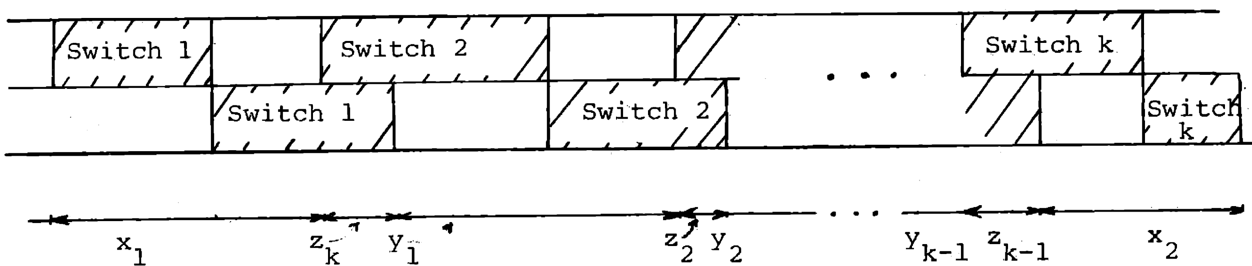


Figure 32

First we can argue that if $\mu_1(J) \neq \overline{\mu}_1(J)$ the pattern in Figure 30c is impossible. This is because in order for every inversion to introduce idle time $\beta_2 > \alpha_4$ and $\alpha_4 > \beta_2$. Therefore the only pattern of switches we have not examined is that in Figure 30d (or Figure 32).

Let K be the maximum number of interacting switches we can argue that:

K	$\frac{\mu_1(J)}{\overline{\mu}_1(J)} \leq$
2	4/3
3	11/8
4	14.3/10
K	3/2

Let I be the idle time introduced by the switches, using the lemma and the notation of figure 32 we have

(a) For K=2

$$I \leq \min(\tfrac{1}{4}(x_1+z_1) + \tfrac{1}{4}(x_2+z_1),\ \tfrac{1}{2}x_1 + \tfrac{1}{4}(x_2+z_1),\ \tfrac{1}{2}x_2 + \tfrac{1}{4}(x_1+z_1))$$

$$\leq \tfrac{1}{3}(x_1 + z_1 + x_2)$$

(b) For K=3

$$I \leq \min(\overbrace{\tfrac{1}{4}(x_1+z_1) + \tfrac{1}{4}(z_1+y_1+z_2) + \tfrac{1}{2}x_2}^{A},\ \overbrace{\tfrac{1}{2}x_1 + \tfrac{1}{4}(z_1+y_1+z_2) + \tfrac{1}{4}(x_2+z_2)}^{B}$$

$$\leq \tfrac{3}{8}(x_1+z_1+y_1+z_2+x_2)$$

This is because if $A < B \quad z_1+x_2<x_1+z_2$

$$A = \tfrac{1}{4}(x_1+z_1) + \tfrac{1}{4}(z_1+y_1+z_2) + \tfrac{1}{2}x_2 \leq \tfrac{3}{8}(x_1+z_1+y_1+z_2+x_2)$$

A symmetric argument holds for $A \geq B$

(c) For K=4, let I_i be the idle time introduced by switch i.

$$I_1 \leq \frac{1}{4}(x_1+z_1)$$

$$I_2 \leq \frac{1}{4}(z_1+y_1+z_2)$$

$$I_3 \leq \frac{1}{2}(y_2+z_3)$$

$$I_4 \leq \frac{1}{4}(z_2+z_3)$$

$$I_1 + I_2 \leq \frac{1}{3}(x_1+z_1+y_1+z_2)$$

$$I_2 + I_3 \leq \frac{1}{3}(z_1+y_1+z_2+y_2+z_3)$$

$$I_3 + I_4 \leq \frac{1}{3}(z_2+y_2+z_3+x_2)$$

$$I_1 + I_2 + I_3 \leq \frac{3}{8}(x_1+z_1+y_1+z_2+y_2+z_3)$$

$$I_2 + I_3 + I_4 \leq \frac{3}{8}(z_1+y_1+z_2+y_2+z_3+x_2)$$

$$2I_2 \leq x_1$$

$$2I_4 \leq x_2$$

$$5I = 5(I_1+I_2+I_3+I_4) \leq x_1(\frac{1}{4} + \frac{1}{3} + \frac{3}{8} + 1) + z_1(\frac{1}{2} + \frac{2}{3} + \frac{3}{4}) +$$

$$+ y_1(\frac{1}{4} + \frac{2}{3} + \frac{3}{4}) + z_2(\frac{1}{4} + \frac{3}{3} + \frac{3}{4}) +$$

$$+ y_2(\frac{1}{2} + \frac{2}{3} + \frac{3}{4}) + z_3(\frac{3}{4} + \frac{2}{3} + \frac{3}{4}) +$$

$$+ x_2(\frac{1}{4} + \frac{1}{3} + \frac{3}{8} + 1)$$

$$< 2.15(x_1 + z_1 + y_1 + z_1 + y_2 + z_3 + x_1)$$

(d) For K arbitrary it is trivial to show the 3/2 bound.

Although in this approach we do not look at what actually happens in each switch deriving bounds is quite tedious. Also this approach cannot give us the 5/4 bound.

By looking at interacting switches macroscopically (that is ignoring the α's and β's) we are actually loosing a lot, because when the switches interact the idle time inserted by two inversions might be less than the sum of the idle times inserted if the switches did not interact. Also we can combine pairs (α_i, β_i) when we put them at the end of the schedule.

From the pattern in Figure 30d we can get permutation schedules by using the heuristics of the lemma. We have to pick the minimum of the following functions and see how it compares to the length of the switches.

What we wish to prove is that

If $I = \min(\alpha_2-\alpha_3 + \beta_5-\beta_6$

$\alpha_2-\alpha_3 + \max(\alpha_6, \beta_6)$

$\alpha_2-\alpha_3 + \alpha_5-\beta_4$

$\max(\alpha_3, \beta_3) + \beta_5-\beta_6$

$\max(\alpha_3, \beta_3) + \max(\alpha_6, \beta_6) - \max(\min(\alpha_3, \beta_6), \min(\alpha_6, \beta_3))$

$\max(\alpha_3, \beta_3) + \alpha_5-\beta_4$

$\max(\beta_2-\alpha_4, \alpha_5-\beta_4)$ (here the interaction of switches results in savings)

$\beta_2-\alpha_4 + \max(\alpha_6, \beta_6)$

$\beta_2-\alpha_4 + \beta_5-\beta_6)$

for
$$\left.\begin{array}{l} \alpha_2 > \beta_1 \geq \alpha_3 \\ \beta_2 > \alpha_4 \geq \beta_3 \\ \alpha_5 > \beta_4 \geq \alpha_6 \\ \beta_5 > \alpha_7 \geq \beta_6 \end{array}\right\}$$

$$I \leq \frac{1}{4} d \quad \text{where } d = \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 + \beta_5 + \beta_6 =$$

$$= \alpha_2 + \alpha_3 + \beta_3 + \beta_2 + \beta_4 + \beta_5 + \beta_6$$

Even a case by case analysis, which would be extremely tedious would give only a partial result. Yet, because of the larger number of permutation schedules we minimize over, the 5/4 bound seems to hold.

<u>Conjecture</u> $\frac{\mu_1(J)}{\bar{\mu}_1(J)} \leq \frac{5}{4}$

In fact since this analysis is only taking local modifications into account the actual bound must be much closer to $\frac{16}{15}$.

4. THE ASYMMETRIC TRAVELING SALESMAN PROBLEM

4.1 Neighborhood Search Technique and Primary Changes

The Traveling Salesman Problem (TSP) (dating at least since 1930) is formulated as follows. "Given a complete directed graph G = (V,E) and a weight c_{ij} for each arc (i,j) in E find a Hamilton circuit (tour) of G of minimum total weight". A comprehensive survey up to 1968 is in [2]. In general the TSP is asymmetric and we denote this as A-TSP. Particular cases result from symmetry, triangle inequality, distance norms between nodes (i.e., Euclidean) and the Gilmore Gomory distance function.

There are various integer programming formulations of the problem. It is related both to the linear assignment problem and the minimum spanning tree (arborescence for directed graphs), which are easy to solve and provide lower bounds for the optimal tour. The general solution procedures can be divided into Tour Building (such as application of dynamic programming or certain heuristics such as insertion methods [35]), Subtour Elimination (the assignment problem is solved and subtours are eliminated from its solution), Use of Spanning Tree Bounds (such as the work of Held and Karp [19], and Tour-to-Tour improvement (neighborhood search techniques of which the Lin Kernighan [25] algorithm is the most successful).

The TSP is NP-complete so is the triangle inequality TSP since it is proven that the Euclidean TSP [30] is NP-complete. [38] and [28] contain results on the complexity of local search and on NP-completeness of having ε-approximate solutions, i.e., $\frac{T_{alg.} - T_{opt.}}{T_{opt.}} \leq \varepsilon$. [29] justifies

these results by the construction of difficult TSP examples. Yet the problem of ε-approximate algorithms for triangle inequality TSP's is open. In fact [4] and [35] give such algorithms for $\varepsilon=1, \frac{1}{2}$ in the symmetric case.

The underlying theme of the preceeding chapters was that certain flowshop scheduling problems are hard, but can be approximated because their structure is close to a special Asymmetric Traveling Salesman Problem. Unfortunately, even if the distances of the TSP have a lot of structure (as in the (4,0)-FS case) the problem is still untractable. Three are the significant features of these TSP's.

a) They are highly asymmetric

b) They satisfy the triangle inequality (Section 1.2)

c) $\frac{T_R}{T_{opt}} \leq m$ (T_R: cost of random tour) for the (m,0)-FS problem (section 3.3)

The triangle inequality alone (without symmetry) does not seem to guarantee that the ε-approximate algorithms for the symmetric case work as well for the asymmetric case. For the general A-TSP most approaches are of the branch and bound type with intelligent lower bounds [24]. There is a lack of heuristics similar to [25]. In order to see how these successful heuristics can be applied to the A-TSP we will study the notions of neighborhood structure and primary changes for the asymmetric case.

Let us use the framework of [28], denoting the set of nonnegative integers by Z^+ and $[n] = \{1,2,\ldots n\}$.

Definition: A combinatorial optimization problem with numerical input (COPNI) is a pair (n,F), where $n \in Z^+$ is the dimension of the problem and F, a subset of $2^{[n]}$, is the set of feasible solutions. We will require that there exists at least one feasible solution and that no feasible solution is properly contained in another.

An instance of the COPN1 (n,F) is a function (numerical input) $c: [n] \to Z^+$. In order to solve an instance of the COPN1 (n,F) we are required to find a feasible solution $f \in F$ such that $c(f) = \sum_{j \in f} c(j)$ is minimum.

The A-TSP is an example of a COPN1 where for τ nodes we have $n = \tau(\tau-1)$ and F the set of all possible tours represented as sets of τ links.

In this general setting we informally wish to assign to each feasible solution its neighborhood $N(f)$ (a set of other feasible solutions).

Definition: A neighborhood structure for the COPN1 (n,F) is a function $N: F \to 2^F$.

The purpose of the above machinery is to make exact the notion of neighborhood of a feasible solution in various combinatorial problems and to explain how local search algorithms for the COPN1 (n,F) and the neighborhood structure N work.

These algorithms determine local optima starting by a random solution and improving it by neighborhood search. The heart of these heuristics is the function IMPROVE (f,c), which when invoked, returns some $s \in N(f)$ such that $c(s) < c(f)$, if such an s exists, and returns

'no' otherwise.

$f := f_0$

while IMPROVE (f,c) $\neq$ "no" do

$f := \text{IMPROVE}(f,c)$

return f

The output is a local optimum with respect to N for the instance c of (n,F). The neighborhood structure affects the complexity of IMPROVE, the number of executions of the while loop, and the quality of the local optima. N is exact if all local optima are also global optima.

The minimal exact neighborhood based on [38] has a nice characterization, which unfortunately is not algorithmic oriented and does not lend itself to anything but exhaustive search in the general A-TSP case.

Theorem 4.1: In a COPN1 (n,F) there exists a unique minimal exact neighborhood structure given by

$$\hat{N}(f) = \{s \in F: \text{for some instance } c, s \text{ is uniquely optimal with } f \text{ second to optimal}\}$$

It should be noted that the complexity characterization of [28] for neighborhood structures of TSP's makes it highly improbable to achieve ε-approximate solutions (for all $\varepsilon>0$) via neighborhood search, but it does not include the triangle inequality restriction.

The successful Lin Kernighan algorithm is based on an intelligent (although unfortunately exponential) search of a subset of the minimal exact neighborhood. The changes it makes in order to IMPROVE a solution

are primary-changes and as [38] shows the set of primary changes is a subset of the $\hat{N}(f)$. We will study the consequences of the asymmetric assumption on the primary changes and latter on base our heuristic on these consequences.

PRIMARY CHANGES

Definition 1: If f and f' are two A-TSP solutions such that f' can be produced from f by exchanging λ links in f with λ links not in f, we say that f and f' are λ-changes of one another.

Given a feasible A-TSP solution f, let $A = \{a_1, a_2, \ldots a_\lambda\}$ be a set of links belonging to f and $B = \{b_1, b_2, \ldots b_\lambda\}$ a set of links not belonging to f. We dnote by G(A,B) the graph whose vertex set corresponds to the set of links A. Let $a_i = (\alpha, \beta)$ and $a_j = (\gamma, \delta)$. A directed edge b connects i and j in G(A,B) iff b is in B and of the form (α, δ). f' = f-A+B denotes an interchange of links A with links B.

Lemma 4.1.2: If f' = f-A+B is an λ-change of f then G(A,B) consists of λ edges forming one or more disjoint cycles.
(The difference from [38] is that we no longer restrict ourselves to special nonadjacent sets of links A).

Proof: Assume f' is an λ-change of f and $b \in B$ is the link between nodes α and δ. In order that these nodes end up with exactly two directed incident links a link going out of α and into δ must be removed. Thus we have each end of b adjacent to a link of A and the ingoing-outgoing direction constraints imply that each of the λ

b's appears in G. Now assume that a_i links α and β, since in a feasible solution every city must have exactly one ingoing and one outgoing link B must contain exactly one link outgoing edge (α, δ_1) and one ingoing edge (γ_2, β). Thus each vertex of G has degree 2 and G consists of disjoint cycles (Using the ingoing outgoing conventions we are able to include cases where an a_i and a_j are adjacent).

Definition 2: If f' is an λ -change of f such that G(A,B) consists of a single cycle then f' is a primary change of f.

The theorem 2 of [38] is invariant to the A-TSP and shows that primary changes are a subset of the $\hat{N}(f)$ and therefore good candidates for a search.

Theorem 4.1.3: If f, f' feasible ATSP solutions and f' a primary λ-change of f then $f' \in \hat{N}(f)$.

Let us now count the number of possible λ-changes.

Theorem 4.1.4: Let f be a feasible A-TSP solution on n nodes then it has $\binom{n}{\lambda}\sum_{i=0}^{\lambda-1} (-1)^i \binom{\lambda}{i} (\lambda-i-1)! + (-1)^\lambda$ possible λ-changes.

Proof: There are $\binom{n}{\lambda}$ possible choices for deleting λ-edges and in the directed case it does not matter if they are adjacent (in the directed case it does matter because: for a directed tour we should always enter at A and exit at B, as in figure 33, therefore the part of the tour AB may be regarded as a point . In the undirected case if $A \neq B$ we have two choices but if A=B we have only one).

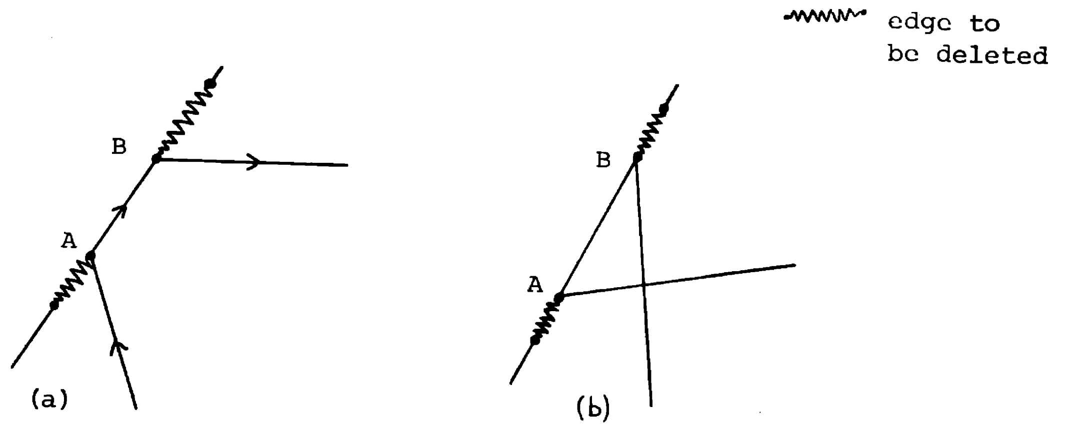


Figure 33

(a) unique choice of directed case
(b) extra choice for undirected case A≠B

The counting problem in the directed case is the following: How many cycles of the integers modulo λ are there such that a_i is never followed by $(a_i+1) \bmod \lambda$. (These cycles correspond to the f' that result from fixing λ edges in f and replacing them in an λ-change fashion . The a_i's of the problem above correspond to the circled numbers in the Figures 34a and b).

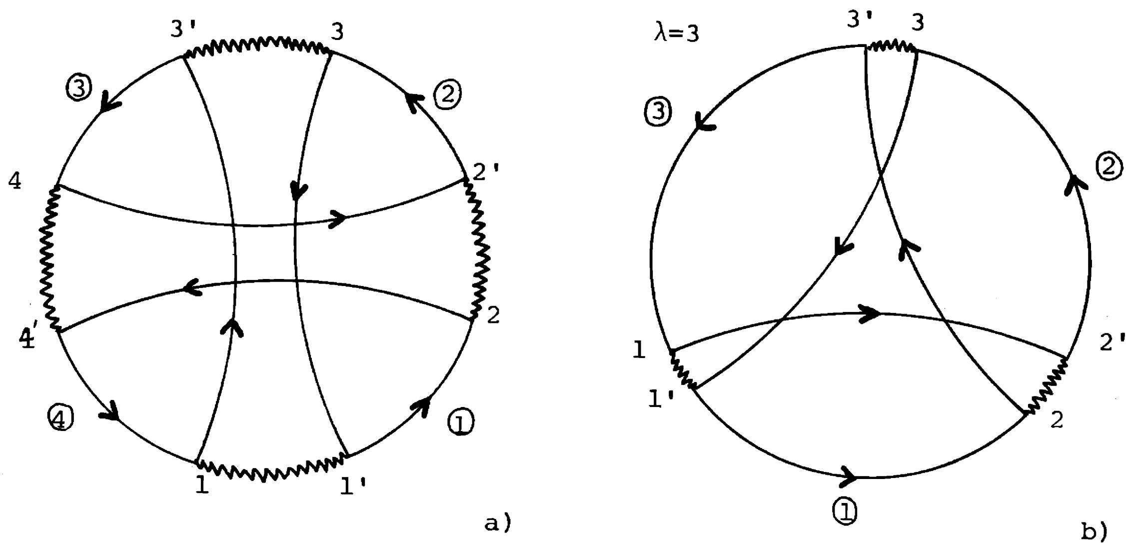


Figure 34

Let P_i be the property that $i \to i+1 \bmod \lambda$ appears in a permutation. Whenever we choose k properties this divides $\{1,2,\ldots\lambda\}$ into $\lambda-k$ blocks and the elements of a block consist of $(a, a+1,\ldots a+\ell)$ for some a and ℓ. e.g., $\lambda=6$ and $P_1, P_2, P_4 \to k=3$ (123) (45) (6) three blocks.

It is clear that the number of cycles satisfying the k properties is the number of cycles on the blocks $(\lambda-k-1)!$ This holds except when $\lambda=k$ in which case there is one block and one cycle. By inclusion exclusion the counting gives us

$$A(\lambda) = \sum_{i=0}^{\lambda-1} (-1)^i \binom{\lambda}{i} (\lambda-i-1)! + (-1)^\lambda$$

λ	$A(\lambda)$
2	0
3	1
4	1
5	8
6	36
7	229
8	1625
9	13208
10	120288
11	1214673

So in the directed case the full solution is $\binom{n}{\lambda} A(\lambda)$ λ-changes.

Note: In the undirected case the factor $\binom{n}{\lambda}$ changes, because of the possibility of deleting adjacent edges. If $[n] = \{1,2,\ldots,n\}$ and $\alpha(n,\times,\lambda)$ is the number of λ-subsets containing $\times$ pairs of adjacent mod λ numbers we have

$$\sum_{\times} \alpha(n,\times,\lambda) = \binom{n}{\lambda}$$

and the possible λ-changes are $\sum_{x} \alpha(n,x,\lambda)\ 2^{\lambda-x} \cdot A(\lambda)$. For non-adjacent changes $\alpha(n,0,\lambda) = \frac{n}{n-\lambda}\binom{n-\lambda}{\lambda}$ [23].

The above λ-changes contain primary and non-primary changes. As in the undirected case the primary changes must form a set of at least exponential size. For this we do not give a proof but we indicate that the technique of [38] for the construction of primary λ-changes is no longer valid.

In the undirected case "For every distinct cycle on the λ-set of edges that we delete $\{a_1, a_2, \ldots a_\lambda\}$ there is a primary λ-change". In the directed case there might be none. For $\lambda=4$ there is only one change and it is not primary yet we can have many cycles in 4 points.

In the undirected case there can be more than one primary changes to a cycle on the λ-set of edges (Figure 35).

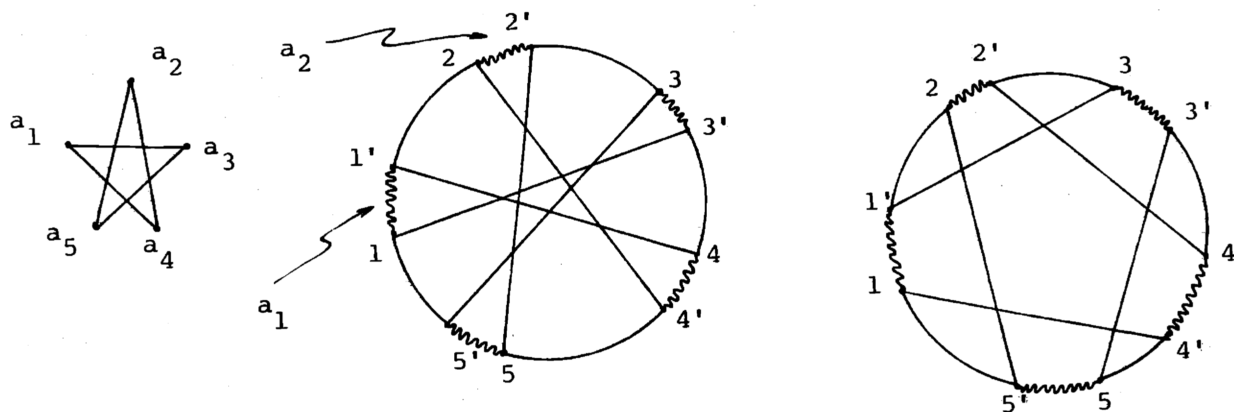


Figure 35

In the directed case there are $(\lambda-1)!$ cycles on $\{a_1,\ldots a_\lambda\}$. Part of them correspond to primary changes. Let us now observe an important property of that part.

Lemma 4.1.4: All primary λ-changes in the directed case are odd-changes.

Proof:

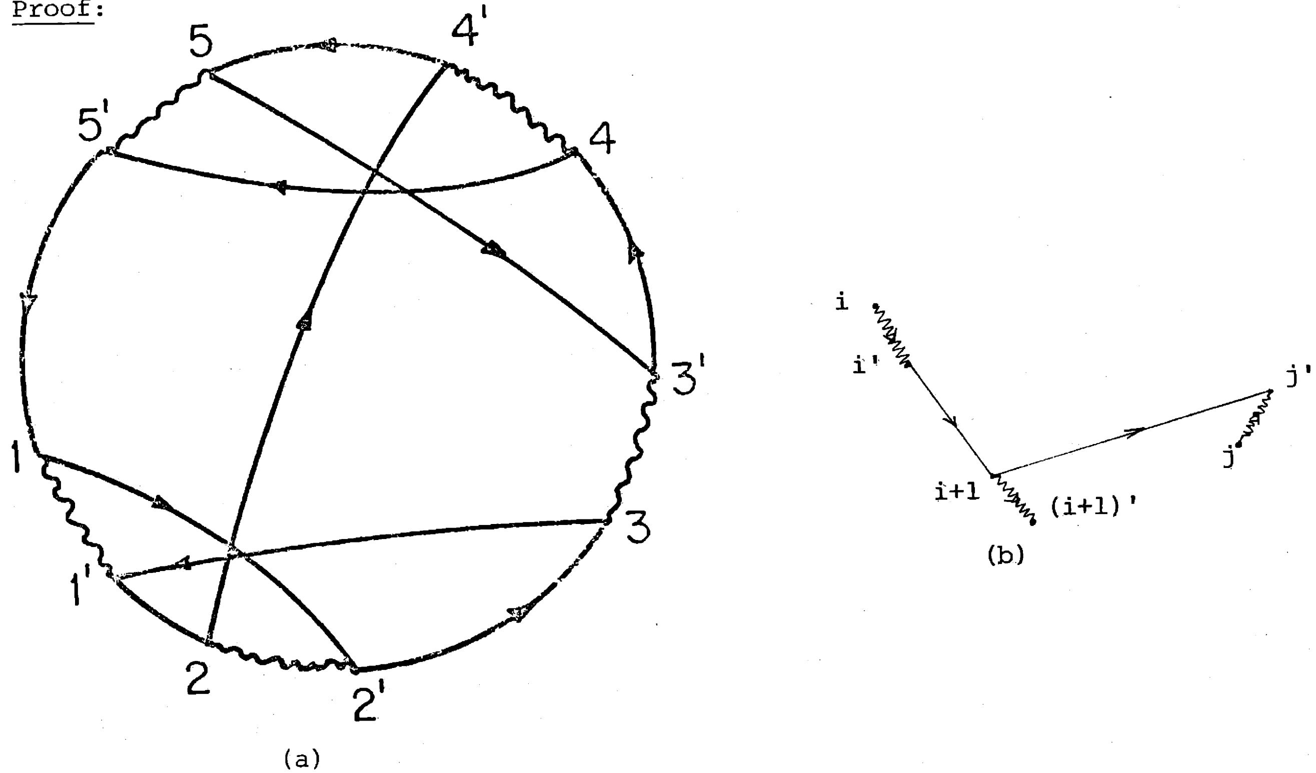


Figure 36

Suppose we have a primary even change. Then we would have, using the obvious notation for the nodes of figure 36a and b, three cyclic permutation of the λ points $(1,2,3,\ldots\lambda)$

$\pi_f : i \to i+1$ (initial tour)

$\pi_f' : i \to j$ (new tour)

$\pi_G : i+1 \to j$ (cycle in graph G(A,B))

Therefore $\pi_{f'} = \pi_G \pi_f$. Each one consists of an odd number of transpositions $\lambda-1$ because it is a cycle. Yet by the above product for π_f, we have (odd + odd) = (even) number of transpositions. This is a contraction.

In closing this section we can note that by exhaustively examining all 5-changes (there are 8 of them) we have that all 5-changes are primary. (This is expected because there is no 2-change).

4.2 The Significance of the Triangle Inequality

As was pointed out in section 4.1 the complexity of ε-approximate algorithms for the A-TSP with the triangle inequality is an open question. In order to justify the complexity of the approximation problem without the triangle inequality [29] contains certain hard examples. These examples serve to trap general purpose neighborhood searches in local suboptimal solutions. The principle "vice" of these examples seems to be the fact that the first and second to optimal solutions (of which there are many) can differ by arbitrarily large amounts. It is not possible to create such cases for the symmetric triangle inequality TSP [29]. In fact we will prove that it is not symmetry but the triangle inequality that is responsible for it.

Theorem 4.2.1: Given an instance of the triangle inequality A-TSP on n nodes. Let T_{opt} be the cost of the optimal tour and T_{sopt} be the cost of the second optimal tour then:

$$\frac{T_{sopt} - T_{opt}}{T_{opt}} \leq \frac{4}{n}$$

Proof: Consider the optimal tour T_{opt} in Figure 36 (a,b,c, path p,d), and the suboptimal tour T_1 (e,b,f,g, path p)

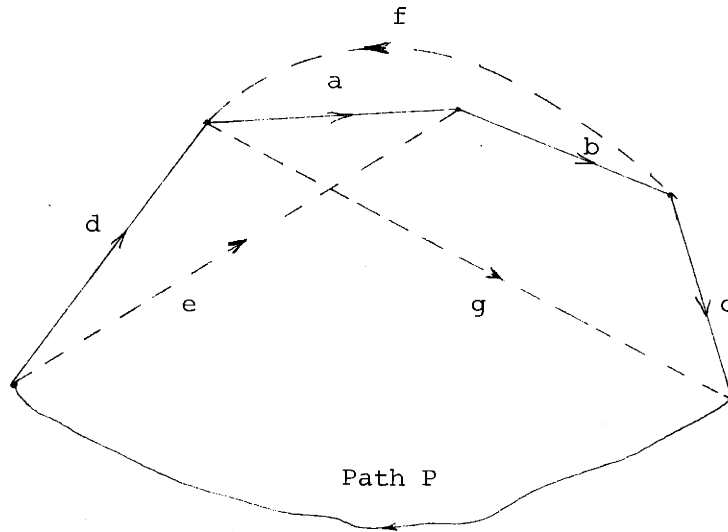


Figure 36

$$\left.\begin{array}{l} e \leq a+d \\ f \leq a+b \\ g \leq a+b+c \end{array}\right\} \Longrightarrow \begin{array}{l} e-d \leq a \\ f-a \leq b \\ g-c \leq a+b \end{array}$$

$$T_1 - T_{opt} = e+f+g - (a+d+c) \leq 2(a+b)$$

$$\frac{T_{sopt} - T_{opt}}{T_{opt}} \leq \frac{T_1 - T_{opt}}{T_{opt}} \leq \frac{2(a+b)}{T_{opt}}$$

Let a,b be the two consecutive edges with least sum. If n=2k then $T_{opt} \geq K(a+b)$, if n=2K+1 then $(2K+1)(a+b) \leq 2\,T_{opt}$ therefore $T_{opt} \geq \frac{n}{2}(a+b)$.

As a consequence $\frac{T_{sopt}-T_{opt}}{T_{opt}} \leq \frac{2}{\frac{n}{2}} = \frac{4}{n}$. □

Another consequence of the triangle inequality is that for the nearest neighbor heuristic,

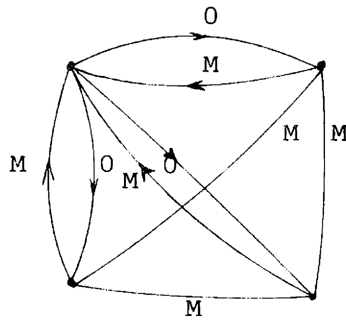
1. Start with an arbitrary node.
2. Find the node not yet on the path, which is closest to the node last added and add to the path the edge connecting these two nodes.
3. When all nodes have been added to the path, add an edge connecting the starting node and the last node added.

We have the bound

$$\frac{T_{near\ neighbor}}{T_{optimal}} \leq \frac{1}{2} \lceil \log(n) \rceil + 1$$

(instead of $\frac{1}{2}\lceil \log(n) \rceil + \frac{1}{2}$ for the symmetric case [35]). The proof of this fact is identical to the proof of [35] with only minor modifications.

We should note that without symmetry the proofs for the ε-approximate bounds do not hold. In the nearest insertion case the proof is actually based on an upper bound on the cost by two trees and in the Christofides algorithm case the bound on the use of a min-spanning tree and a matching. Let C_H be the cost of the cheapest Hamilton path and C_T the cost of the min spanning tree. In the symmetric case $\frac{C_H}{C_T} \leq 2$. (Consider the tree twice and create an Euler path that is transformed to a Hamilton path via shortcuts). In the asymmetric case $\sup \frac{C_H}{C_T} = +\infty$ (*) .



$$\frac{C_H}{C_T} \to \infty$$

Figure 37

Let us propose an extension of the algorithm in [4] for the asymmetric case that combines min spanning arborescence and bipartite matching:

Step 1 Find min spanning 1-arborescence (arborescence containing one circle through the root).

Step 2 S: = {set of leaves}
T: = {i-1 copies of nodes whose outdegree is i>0 in the tree}

Step 3 Find the optimum weighted bipartite matching in (S,T,SxT). (The weights are the same as in the original graph).

Step 4 Now the network contains an Euler tour of at most 2n-2 edges. Make shortcuts using the triangle inequality to make the Euler tour into a Hamilton tour.

Because of (*) above there seems no way to derive analytic upper bounds so it is best to turn our attention to effective heuristics.

4.3 A Proposed Heuristic

In this section we will describe the Lin Kernighan [25] algorithm and the proposed extension to the asymmetric case. Actually the Lin Kernighan algorithm starts from a random tour and makes λ-changes (primary ones basically), such that the sequence of exchanges gives a positive gain and λ is not a fixed number but grows as long as there seems to be potential for larger changes. Four are the strong points of the algorithm:

a) The gain criterion (that says when the λ-change should go on to become an $\lambda+1$-change).

b) The feasibility restriction (that makes it possible after every new exchange to close the tour)

c) The stopping criterion (that stops the λ-changes judging that the algorithm has reached the point of diminishing returns).

d) A limited backtracking facility (backtracking at small levels 2-or 3-changes, based on extensive experimentation).

Lin Kernighan Algorithm [25]

Step 1 Generate a random starting tour T.

Step 2 Set $G^*=0$ [G^* is the best improvement made so far]. Choose any edge x_1, with endpoints t_1 and t_2. Let $i=1$ [i is the level number. Define $g_i = |x_i|-|y_i|$, $|\cdot|$ denotes length]

Step 3 From the endpoint t_2 of x_1, choose edge y_1 to t_3 such that $G_1=g_1>0$. [This is the first application of the gain criterion]. If no such y_1 exists go to Step 6(d) which involves backtracking.

Step 4 Let $i=i+1$. Choose x_i [which currently joins t_{2i-1} and t_{2i}] and y_i as follows:

a) x_i is chosen so that, if t_{2i} is joined to t_1, the resulting configuration is a tour. [Thus for a given y_{i-1}, x_i is uniquely determined. This is the application of the feasibility criterion; it guarantees that we can always "close up" to a tour if we wish, simply by joing t_{2i} and t_1, for any $i \geq 2$. The choice of y_{i-1}, Step 4(e), ensures that there is always such an x_i and so does any choice of y_1]

b) y_i is some available link at the endpoint t_{2i} shared with x_i, subject to (c), (d) and (e). If no y_i exists, go to Step 5 [clearly to make a large cost reduction at the ith step $|y_i|$ should be small, and so in general the choice is based on nearest neighbors. But lookahead can also be implemented and decisions based on $|x_{i+1}| - |y_i|$].

c) x_i cannot be a link previously joined (i.e., a y_j, $j<i$) and similarly y_i cannot be a link previously broken.

d) $G_i = G_{i-1} + g_i > 0$ [Gain criterion]

e) In order to ensure feasibility at $i+1$ y_i must be chosen to permit the breaking of an x_{i+1}.

f) Before y_i is constructed, we check if closing up by joining t_{2i} and t_1 will give a gain value better than the best seen previously. [Since we have satisfied the feasibility criterion for $i \geq 2$ we know this results in a tour]. Let y_i^* be a link connecting t_{2i} and t_1 and let $g_i^* = |y_i^*| - |x_i|$.

If $G_{i-1}+g_i^* > G^*$ set $G^* = G_{i-1}+g_i^*$ and let $K=i$ [G^* is always the best improvement in T recorded so far, and is thus our standard for comparison. $G^* \geq 0$ and monotone nondecreasing. The index K defines the sets to be exchanged to achieve G^*.]

Step 5 Terminate the construction of x_i and y_i in steps 2 through 4 when either no further links x_i and y_i satisfy 4(c) - (e), or when $G_i \leq G^*$. [This is the stopping criterion]. If $G^* > 0$ take the tour T' with $c(T') = c(T) - G^*$ and repeat the whole process from step 2, using T' as the initial tour.

Step 6 If $G^* = 0$, a limited backtracing facility is invoked, as follows:

(a) Repeat steps 4 and 5, choosing y_2's in order of increasing length, as long as they satisfy the gain criterion $g_1 + g_2 > 0$ [If an improvement is found at any time, of course, this causes a return to Step 2].

(b) If all choices of y_2 in Step 4(b) are exhausted without profit, return to Step 4(a) and try to alternate choice for x_2. [This is the only case we violate feasibility and look for a non-primary case, it is rather an ad hoc feature].

(c) If this also fails to give improvement, a backup is performed to Step 3, where the y_1's are examined in order of increasing length.

(d) If the y_1's are also exhausted we try the other x with endpoint t_1 in Step 2.

(e) If this fails we select a new x_1 and repeat a Step 2. [Note that backtracking is performed only if no gain can be found, and only at levels 1 and 2.]

Step 7 The procedure terminates when all links x_1 have been examined without profit [Special "checkout tricks" are used to reduce checkout time of a local optimum].

For the extension of the Lin Kernighan to the Asymmetric case we will follow Lemma 4.1.4. We will not examine even changes, but only odd changes. Thus once more we attempt to search a subset of the minimal exact neighborhood.

We proceed by tentatively adding two new edges every time with a gain criterion

$$G_i = G_{i-2} + g_{i-1} + g_i > 0, \qquad i \text{ even}$$

and a stopping criterion

$$G_i \leq G^*, \qquad i \text{ even}$$

What is very easy to maintain is feasibility because we successively create and break subtours in our tour.

Proposed Heuristic for A-TSP

Step 1 Generate a random tour T.

Step 2 Set $G^* = 0$ [G^* is the best improvement made so far]. Choose any edge $x_1 = (t_1, t_1')$. Let i=1 [i level number, $g_i = |x_i| - |y_i|$].

Step 3 From the endpoint t_1 of x_1, choose edge y_1 to t_2'. $x_2 = (t_2, t_2')$ will then be an edge of the original tour we will have to replace.
Let i=2, from the endpoint t_2, choose an edge y_2 to t_3' such that if $x_3 = (t_3, t_3')$ will be the edge to be deleted then (t_3, t_1) makes the change feasible. [Here we actually created a subtour in the tour and broke it]. Also we must have $G_2 = g_1 + g_2 > 0$ [Gain

criterion]. If no two such y_1, y_2 exist go to step 6(a), which involves backtracking.

Step 4(a) Let i = i+1. Since we know y_{i-1}, we know which x_i has to be deleted from the original tour: $x_i = (t_i, t_i')$. We have picked y_{i-1} so that we can break x_i (step 4(e)). We pick y_i so that we can break x_{i+1} (step 4(e)). Let i = i+1. Again we know which x_i has to be deleted and we pick y_i so that we can break x_{i+1} (step 4(e)). But also so that this y_i goes to t_{i+1}' such that (t_{i+1}, t_1) makes the change feasible. [Here again we break the sub-tour created previously].

(b) The sequences of xyxy that we picked must be subject to (c), (d), (e). If they do not exist go to Step 5. [To make the largest cost reduction we might look at the smallest sum of the y's or we might use something equivalent to the lookahead].

(c) The x's cannot be links previously joined and similarly the y's cannot be links previously broken.

(d) $G_i = G_{i-2} + g_{i-1} + g_i > 0$ [Gain criterion]

(e) The y_i's must be chosen to permit the breaking of the necessary x_{i+1}'s.

(f) Before we construct this change we check if closing the tour before the addition of the last two edges will give a gain value better than the best seen previously. Let $|y^*|$ be the link closing the tour $g^* = |y^*| - x_{i-1}$. If $G_{i-2} + g^* > G^*$ set $G^* = G_{i-2} + g^*$, K=i-1. (The same hold as in 4f of the Lin Kernighan.

Step 5 Terminate the construction of x_i and y_i in steps 2 through 4 when either no further links x_i and y_i satisfy 4(c)-(e), or when $G_i \leq G^*$. [This is the stopping criterion]. If $G^* > 0$ take the tour T' with c(T') = c(T) - G* and repeat the whole process

from step 2, using T' as the initial tour.

Step 6 If $G^* = 0$, a limited backtracking facility is invoked as follows:
[This is different than the Lin Kernighan since what the Link Kernighan achieves by backtracking is following a 3-change approach].

(a) We go back to Step 3 and pick two other y_1, y_2 as long as they satisfy the gain criterion and the feasibility. [If we hit upon an improvement we go to step 2].

(b) We might insert here the possibility to check 4-changes, which are non-primary.

(c) If the above fail we try another x_1.

Step 7 The procedure terminates when all edges x_1 have been examined without profit.

This heuristic is an extension of the Lin Kernighan algorithm and we conjecture that it will be a fairly accurate neighborhood search technique. We should note that the simple fashion that we use to maintain feasibility (creating one subtour and breaking it) does not allow us to reach all primary changes in one iteration (because there are primary changes, where we must create more than one subtour and break them). Trying to extend the algorithm to acquire this capability of choosing many edges while still infeasible would certainly introduce much overhead.

This heuristic could be well suited to the (m,0)-FS problems of the previous chapter.

5. CONCLUSIONS AND OPEN PROBLEMS

Motivated by the fundamental problems of the interaction between buffers size and schedule length in a highly automated flexible manufacturing environment, we have attempted to study the complexity of flowshop problems under buffer constraints.

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 have developed 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 [9] for small b.

We have resolved the status of the (m,0)-FS for $m \geq 4$. Our results of Sections 2.2, 2.3 leave only one open question, as far as no-wait problems are concerned: the 3-machine case. 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 such that $|K| = 1$.

The formalism in [14], suggests that the 1-buffer 2 machine flowshop problem is, like the TSP and 3MI , strongly NP-complete; that is, unless

$P = NP$ 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 2.2, 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 2.2 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 [25] 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 [4]. We develop a methodology for asymmetric TSP's paralleling that of [25], so as to cope with the intricate pecularities of the asymmetric case.

Section 3.4 is devoted to the special subject of Permutation Scheduling. We conjecture that removing the FIFO assumption results in small gains, at least for the b=1 case. For the (2,1)-FS problem we have left open the question of the $\frac{5}{4}$ bound.

Finally, we would like in closing this chapter to point out once again the significance of flowshop problems as those addressing the simplest network of machines. Although we fear that many questions about them have remained unanswered we hope to have clarified some of the issues.

APPENDIX

This Appendix contains a listing of an implementation of the Gilmore-Gomory Algorithm in LISP (pp.111-115). The programs used in the simulation for the (2,b)-FS problem are also listed (pp.116-118).

```
(COMMENT THIS IS THE GILMORE GOMORY ALGORITHM)

(COMMENT THE ALGORITHM IS WRITTEN IN STAGES [ P1- T4] A FUNCTION
         CORRESPONDS TO EACH STAGE- ITS INPUT IS THE ORIGINAL
         NONEMPTY LIST OF JOBS [[A B] ;] ITS OUTPUT IS THAT OF THE
         ALGORITHM AFTER THE EQUIVALENT STAGE- EACH STAGE IS
         CONSTRUCTED THROUGH AUXILIARY FUNCTIONS IE P1A P1B WHICH
         MIGHT BE USED SUBSEQUENTLY INSTEAD OF IE P1)

(COMMENT P1-ARRANGE THE B 'S IN ORDER OF SIZE AND RENUMBER THE JOBS
         SO THAT WITH THE NEW NUMBERING B[I] LESS THAN B[I+1] I=0
         N-2 THE OUTPUT HAS FORM [[1 A B] ])

(DEFUN LESSPA (X Y) (LESSP (CADR X) (CADR Y)))

(DEFUN P1A (L) (SORT (APPEND L NIL) 'LESSPA))

(DEFUN P1B (L)
       (PROG (LS)
             (DO I
                 0.
                 (1+ I)
                 (NULL L)
                 (SETQ LS
                       (APPEND LS (LIST (LIST I (CAAR L) (CADAR L)))))
                 (SETQ L (CDR L)))
             (RETURN LS)))

(DEFUN P1 (L) (P1B (P1A L)))

(COMMENT P2- ARRANGE THE A 'S IN ORDER OF SIZE)

(DEFUN P2 (L) (SORT (APPEND (P1 L) NIL) 'LESSPA))

(COMMENT P3-FIND F[P] FOR ALL P-THE PERMUTATION F IS DEFINED BY
         F[P]=Q Q BEING SUCH THAT AQ IS THE PTH SMALLEST OF THE A
         'S-THE OUTPUT IS A LIST [[BI AFI] ]-F1 IS AN ARRAY
         CONTAINING THE INITIAL PERMUTATION-F2 CONTAINS THE OUTPUT)

(DEFUN P3 (L)
       (PROG (L1 L2 L3)
             (SETQ NL (LENGTH L))
             (ARRAY F1 T NL)
             (ARRAY F2 T NL)
             (SETQ L1 (P1A L))
             (SETQ L2 (P1B L1))
             (SETQ L3 (SORT (APPEND L2 NIL) 'LESSPA))
             (DO I
                 0.
                 (1+ I)
                 (= I NL)
                 (STORE (F1 I) (LIST (CAAR L3)))
                 (STORE (F2 I) (LIST (CADAR L1) (CADAR L3)))
                 (SETQ L1 (CDR L1))
                 (SETQ L3 (CDR L3)))
             (RETURN (LISTARRAY 'F2))))
```

```
(COMMENT P4-COMPUTE THE NUMBERS C[AII+1] FOR I=0 N-2
         C[AII+1]=MAX[[MIN[BI+1 AFI+1]-MAX[BI AFI]] 0] F3 IS AN
         ARRAY WHERE THE C 'S ARE STORED A LABEL INDICATING WHETHER
         RII+1 BELONGS TO GROUP1 OR 2. F3[I]=[I C K])

(DEFUN P4 (L)
       (PROG (A B C D A1 C1 K)
             (P3 L)
             (ARRAY F3 T (SUB1 NL))
             (DO I
                 0.
                 (1+ I)
                 (= I (SUB1 NL))
                 (SETQ J (1+ I))
                 (SETQ A (CAR (F2 I)))
                 (SETQ B (CADR (F2 I)))
                 (SETQ C (CAR (F2 J)))
                 (SETQ D (CADR (F2 J)))
                 (SETQ A1 (MAX A B))
                 (SETQ C1 (MIN C D))
                 (COND ((EQUAL A1 B) (SETQ K 1.))
                       ((EQUAL A1 A) (SETQ K 2.)))
                 (STORE (F3 I)
                        (LIST I (MAX (DIFFERENCE C1 A1) 0.0) K)))
             (RETURN (LISTARRAY 'F3))))

(COMMENT S1-FORM AN UNDIRECTED GRAPH WITH N NODES AND UNDIRECTED
         ARCS CONNECTING THE ITH AND FTH NODES I=0 N-1 -RL CONTAINS
         THE SORTED [I C K]-F1 CONTAINS PAIRS [FI [LABEL OF
         SUBTOUR]]-K1 IS THE NUMBER OF SUBTOURS IN EXCESS)

(DEFUN S1 (L)
       (PROG (K1)
             (SETQ RL (P4 L))
             (SETQ RL (SORT (APPEND RL NIL) 'LESSPA))
             (SETQ K1 0.)
             (DO I
                 0.
                 (1+ I)
                 (= I NL)
                 (COND ((NUMBERP (CADR (F1 I))))
                       (T (DO ((J I (CAR (F1 J))))
                              ((= (CAR (F1 J)) I)
                               (STORE (F1 J) (LIST (CAR (F1 J)) K1))
                               (SETQ K1 (ADD1 K1)))
                              (STORE (F1 J)
                                     (LIST (CAR (F1 J)) K1))))))
             (RETURN (SUB1 K1))))

(SETQ LIT '((34.0 31.0)
            (45.0 13.0)
            (4.0 3.0)
            (18.0 40.0)
            (22.0 26.0)
            (16.0 15.0)
            (7.0 1.0)))
```

```
(COMMENT THIS LIT IS A USEFUL EXAMPLE S2-IF THE CURRENT GRAPH HAS
         ONLY ONE COMPONENT GO TO STEP T1-OTHERWISE SELECT THE
         SMALLEST VALUE C [AII+1] SUCH THAT I IS IN ONE COMPONENT
         AND I+1 IN ANOTHER-IN CASE OF A TIE FOR SMALLEST CHOOSE
         ANY- S3-ADJOIN THE UNDIRECTED ARC RII+1 TO THE GRAPH USING
         THE I VALUE SELECTED IN S2-RETURN TO S2-T1-DIVIDE THE ARCS
         ADDED IN S3 INTO TWO GROUPS-GROUP1 HAS BI GREATER THAN AFI
         THE REST ARE IN GROUP2-T23-SORT THE ADDED BRANCHES [ I1
         GREATER I2 ETC IN GROUP1][J1 LESS J2 IN GROUP2])

(DEFUN T23 (R1 R2)
       (APPEND (SORT (APPEND R2 NIL) 'GREATERP)
               (SORT (APPEND R1 NIL) 'LESSP)))

(DEFUN S23T1 (L)
       (PROG (R1 R2 X1 X2 X3 Y1 COUNTER1)
             (SETQ COUNTER1 (S1 L))
             (DO ((I COUNTER1 (SUB1 I)))
                 ((= I 0.) (SETQ R12 (T23 R1 R2)))
                LABEL1
                (SETQ X1 (CAAR RL))
                (SETQ X2 (CADR (F1 X1)))
                (SETQ X3 (CADR (F1 (1+ X1))))
                (SETQ Y1 (CADDAR RL))
                (COND ((= X2 X3) (SETQ RL (CDR RL)) (GO LABEL1))
                      (T (SETQ RL (CDR RL))
                         (COND ((= Y1 1.)
                                 (SETQ R1 (APPEND R1 (LIST X1))))
                                ((= Y1 2.)
                                 (SETQ R2 (APPEND R2 (LIST X1))))))
                         (DO J
                             0.
                             (1+ J)
                             (= J NL)
                             (COND ((= (CADR (F1 J)) X3)
                                    (STORE (F1 J)
                                           (LIST (CAR (F1 J)) X2)))
                                   (T))))))))

(COMMENT T4-THE OPTIMAL TOUR IS OBTAINED BY ELIMINATION OF SUBTOURS
         IN F1 USING THE BRANCHES STORED IN R12)
```

```
(DEFUN T4 (L)
       (PROG (Z1 S1 S2 A B)
             (S23T1 L)
             (ARRAY F3 FIXNUM NL)
             (DO I 0. (1+ I) (= I NL) (STORE (F3 I) I))
             (DO I
                 0.
                 (1+ I)
                 (NULL R12)
                 (SETQ Z1 (CAR R12))
                 (SETQ R12 (CDR R12))
                 (DO J
                     0.
                     (1+ J)
                     (= Z1 NL)
                     (COND ((EQUAL (F3 J) Z1) (SETQ S1 J) (SETQ A 1.))
                           ((EQUAL (F3 J) (1+ Z1))
                            (SETQ S2 J)
                            (SETQ B 1.))
                           (T))
                     (COND ((AND (EQUAL A 1.) (EQUAL B 1.))
                            (SETQ Z1 (F3 S1))
                            (STORE (F3 S1) (F3 S2))
                            (STORE (F3 S2) Z1)
                            (SETQ Z1 NL)
                            (SETQ A 0.)
                            (SETQ B 0.))
                           (T))))
             (DO I
                 0.
                 (1+ I)
                 (= I NL)
                 (SETQ S2 (F3 I))
                 (STORE (F3 I) (CAR (F1 S2))))
             (FILLARRAY 'F2 (P1A L))
             (SETQ S1 0.)
             (DO I
                 0.
                 (1+ I)
                 (= I NL)
                 (STORE (F1 I) (F2 S1))
                 (SETQ S1 (F3 S1)))
             (RETURN (LISTARRAY 'F1))))

(COMMENT THIS PART GIVEN A LIST L OUTPUTS ALIST IN THE GILMORE
         GOMORY ORDER- ALSO THE ARRAY DISTANCE CONTAINS THE TSP
         DISTANCES- THESE CORRESPOND TO THE GI-GO SEQUENCE)
```

```
(DEFUN GIGO (L)
       (PROG (OUTPUT)
             (SETQ OUTPUT (T4 L))
             (ARRAY DISTANCE FLONUM NL NL)
             (DO
              I
              0.
              (1+ I)
              (= I NL)
              (DO
               J
               0.
               (1+ J)
               (= J NL)
               (COND ((= I J) (STORE (DISTANCE I J) 0.0))
                     (T (STORE (DISTANCE I J)
                               (MAX 0.0
                                    (DIFFERENCE (CAR (F1 J))
                                                (CADR (F1 I))))))))))
             (RETURN OUTPUT)))
```

```
(DEFUN SCH (L Z TN SN)
       (PROG (LO L1 N K)
             (SETQ N (LENGTH L))
             (SETQ K (PLUS 2. Z))
             (ARRAY TN FLONUM (ADD1 N))
             (ARRAY SN FLONUM (ADD1 N))
             (DO I
                 0.
                 (ADD1 I)
                 (= I (ADD1 N))
                 (COND ((= I 0.)
                        (STORE (SN I) 0.0)
                        (STORE (TN I) 0.0))
                       ((LESSP (SUB1 I) K)
                        (STORE (SN I) (PLUS (SN (SUB1 I)) (CAAR L)))
                        (STORE (TN I)
                               (PLUS (CADAR L)
                                     (MAX (SN I) (TN (SUB1 I)))))
                        (SETQ L (CDR L)))
                       (T (STORE (SN I)
                                 (PLUS (CAAR L)
                                       (MAX (SN (SUB1 I))
                                            (TN (- I K)))))
                          (STORE (TN I)
                                 (PLUS (CADAR L)
                                       (MAX (SN I) (TN (SUB1 I)))))
                          (SETQ L (CDR L)))))
             (SETQ LO (LISTARRAY SN))
             (SETQ L1 (LISTARRAY TN))
             (RETURN (LIST LO L1))))

(DEFUN SCHEDULE (L Z) (SCH L Z 'TN 'SN))

(DEFUN JOHNP (X Y)
       (LESSP (MIN (CAR X) (CADR Y)) (MIN (CAR Y) (CADR X))))

(DEFUN FLOWT (L)
       (COND ((NULL L) 0.0) (T (PLUS (CAR L) (FLOWT (CDR L))))))

(DEFUN MAKESPAN (L Z) (CAR (LAST (CADR (SCHEDULE L Z)))))

(DEFUN FLOWTIME (L Z) (FLOWT (CADR (SCHEDULE L Z))))

(COMMENT GIVEN A LIST OF JOBS L AND BUFFER CAPACITY Z SCHEDULE
         STORES IN TN AND SN THE STARTING TIMES IN THE 2ND AND 1RST
         MACHINES-MAKESPAN AND FLOWTIME CALCULATE THEMSELVES)
```

```
(COMMENT -NRANDOM PRODUCES A RANDOM INTEGER BETWEEN B AND C)

(DEFUN NRANDOM (B C)
       (PROG (X)
             (SETQ X (PLUS C 1. (MINUS B)))
             (SETQ X (RANDOM X))
             (SETQ X (PLUS X B))
             (RETURN X)))

(COMMENT -REXP1[A] MAKES A A FLOATING POINT NUMBER AND DIVIDES BY
         10000.0)

(DEFUN REXP1 (A) (//$ (FLOAT A) 10000.0))

(COMMENT -REXP CONSTRUCTS A SATURATED SCHEDULE ON TWO MACHINE
         FLOWSHOPS WITH ONE BUFFER WE HAVE [A+2] TASKS OF WHICH ONE
         IS [0 0])

(DEFUN REXP (A)
       (PROG (L1 L2 L3 B C D)
             (SETQ OPTIMAL 0.)
             (DO I
                 0.
                 (1+ I)
                 (= I A)
                 (SETQ L1 (APPEND L1 (LIST (RANDOM 10000.))))
                 (SETQ OPTIMAL (PLUS OPTIMAL (CAR (LAST L1)))))
             (SETQ L1 (APPEND L1 (LIST 0.)))
             (SETQ OPTIMAL (REXP1 OPTIMAL))
             (SETQ L2 L1)
             (SETQ B 0.)
             (SETQ L3 (LIST B))
             (SETQ C (CAR L2))
             (SETQ D 0.)
             (DO I
                 0.
                 (1+ I)
                 (= I A)
                 (SETQ D (PLUS D (CAR (LAST L3))))
                 (SETQ L3
                       (APPEND L3
                               (LIST (DIFFERENCE (NRANDOM B C) D))))
                 (SETQ B (PLUS B (CAR L2)))
                 (SETQ L2 (CDR L2))
                 (SETQ C (PLUS C (CAR L2))))
             (SETQ L1 (MAPCAR 'REXP1 L1))
             (SETQ L3 (MAPCAR 'REXP1 L3))
             (SETQ L2 '((0.0 0.0)))
             (DO I
                 0.
                 (1+ I)
                 (= I (1+ A))
                 (SETQ L2 (APPEND L2 (LIST (LIST (CAR L3) (CAR L1)))))
                 (SETQ L1 (CDR L1))
                 (SETQ L3 (CDR L3)))
             (RETURN L2)))
```

```
(COMMENT A2 MUST BE GREATER THAN 1-WE EXAMINE CASES FROM 4. TO
        [A1+3] JOBS IN EACH CASE WE HAVE A2 RANDOM EXAMPLES)

(DEFUN SIMULATION (A1 A2)
       (PROG (L X1 X2 X3 X4)
             (ARRAY WC T A1)
             (ARRAY WCER FLONUM A1)
             (ARRAY MEAN FLONUM A1)
             (ARRAY SDEV FLONUM A1)
             (DO JJ
                 0.
                 (1+ JJ)
                 (= JJ A1)
                 (STORE (WCER JJ) 0.0)
                 (SETQ X3 0.)
                 (SETQ X4 0.)
                 (DO II
                     0.
                     (1+ II)
                     (= II A2)
                     (SETQ L (REXP (PLUS 2. JJ)))
                     (SETQ X1 (MAKESPAN (GIGO L) 1.))
                     (SETQ X2 (QUOTIENT (DIFFERENCE X1 OPTIMAL)
                                        OPTIMAL))
                     (SETQ X3 (PLUS X3 X2))
                     (SETQ X4 (PLUS X4 (TIMES X2 X2)))
                     (COND ((GREATERP X2 (WCER JJ))
                            (STORE (WCER JJ) X2)
                            (STORE (WC JJ) L))
                           (T)))
                 (STORE (MEAN JJ) (QUOTIENT X3 A2))
                 (STORE (SDEV JJ)
                        (SQRT (QUOTIENT (DIFFERENCE X4
                                                    (TIMES X2
                                                           (MEAN JJ)))
                                        (1- A2)))))
             (RETURN (LISTARRAY 'WCER))))

(DEFUN SIMUL (N)
       (QUOTIENT (DIFFERENCE (MAKESPAN (GIGO (REXP N)) 1.) OPTIMAL)
                 OPTIMAL))

(COMMENT A ONE SHOT EXPERIMENT WITH N+2 JOBS)
```

REFERENCES

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

2. M. Bellmore, G.L. Nemhauser, "The Traveling Salesman Problem a Survey", Operat. Res. 16, 538-558 (1968).

3. R.E. Buten, V.Y. Shen, "A Scheduling Model for Computer Systems with Two Classes of Processors", Proc. 1973, Sagamore Computer Comp. on Parallel Processing Springer Verlag, N.Y., 1973, pp. 130-138.

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

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

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

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

8. S.A. Cook, "The Complexity of Theorem Proving Procedures", Proc. 3rd Annual ACM Symposium on Theory of Computing, 151-158.

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

10. 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.

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

12. 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).

13. 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.

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

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

16. 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.

17. T. Gonzales, and S. Sahni, "Flowshop and Jobshop Schedules: Complexity and Approximation", Operat. Res. Vol. 26, No. 1, Jan.-Feb. 1978, pp. 36-52.

18. 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).

19. M. Held, R.M. Karp, "The Traveling Salesman Problems and Minimum Spanning Trees", Operat. Res. 18, (1138-1162) (1970).

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

21. 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.

22. R.M. Karp, "Probabilistic Analysis of Partitioning Algorithms for the Traveling-Salesman Problem in the Plane", Math. of Operations Research, Vol. 2, No. 3, Aug. 77, pp. 209-224.

23. Kaplansky, "Solution of the 'probleme des menages" Bull A.M.S. 49 (1943), pp. 784-5.

24. J.K. Lenstra, "Sequencing by Enumerative Methods", Mathematische Centrum, Amsterdam (1976).

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

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

27. C.H. Papadimitriou, K. Steiglitz, "Combinatorial Optimization Algorithms", book in preparation, 1978.

28. C.H. Papadimitriou, and K. Steiglitz, "The Complexity of Local Search for the TSP", Siam Journal on Computing 6, No. 1, 76-82, (1977).

29. C.H. Papadimitriou, and K. Steiglitz, "Some Difficult Examples of the TSP", Harvard Rep. TR-20-76.

30. C.H. Papadimitriou, "The Euclidean Traveling Salesman Problem is NP-Complete", Journal of Theoretical Computer Science, 1978.

31. C.H. Papadimitriou, P.C. Kanellakis, "Flowshop Scheduling with Limited Temporary Storage", March 1978, ESL-P-808 (also Harvard Univ. TR-05-78).

32. J. Piehler, "Ein Beitrag zum Reihenfolgeproblem"Unternehmensforschung 4, (138-142) (1960)

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

34. R.L. Rivest, private communication.

35. Rosenkrants, Stearns, and Lewis, "Approximate Algorithms for the TSP", 15th SWAT Proc. (1974) (33-42).

36. S. Sahni, Yookum Cho, "Complexity of Scheduling Shops with No Wait in Process", Tech. Report 77-20, Dec. 1977. Univ. of Minnesota.

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

38. Weiner, Savage, and Bagchi, "Neighborhood Search Algorithms for Finding Optimal Traveling Salesman Tours must be Inefficient", 5th SIGACT Proc. (1973), 207-213.

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