SCHEDULING PROBLEMS WITH

RESTRICTED INTERMEDIATE STORAGE

by

NEIL JAMES OTWAY
B.Sc.(Hons.)(Adelaide)

Thesis submitted for the Degree of

Doctor of Philosophy

in the University of Adelaide,

Department of Applied Mathematics.

July, 1980

CONTENTS

SUMMARY

For job-shop processes in which the facilities for storing partially completed jobs have limited capacity, the problem of determining optimal schedules is an area which hitherto has received scant treatment in the literature. The purpose of this thesis is to investigate the intricacies of these problems and to develop a basic framework for their solution.

An interesting difference between the job-shop process which is usually studied and the one discussed here is that when a partially completed job is waiting to be processed by its next machine, a distinction now has to be made between including and excluding the endpoints of the interval representing the waiting time. If such intervals are *closed*, (i.e. both endpoints are included) each job is considered to visit every one of its assigned storage bins. On the other hand, if these intervals are *not* closed, then it is possible to allow for the situation where a job bypasses a storage facility and moves directly to its next machine.

Two specializations of the general model are discussed separately. First, a flowshop process for which there is a storage bin at the end of each machine is considered for the case with closed and half-open waiting intervals respectively. Next, allowing only for closed waiting intervals, a process is investigated which can be interpreted as a train scheduling problem. In both instances a branch and bound solution procedure which avoids duplication of schedules is developed.

Finally, employing some results from the scheduling of jobs on a single machine, methods for determining lower bounds are developed for possible inclusion into the branch and bound procedures for solving the general problem.

SIGNED STATEMENT

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University. To the best of my knowledge and belief, the thesis contains no material previously published or written by any other person, except where due reference is made in the text of the thesis.

Neil J. Otway

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. F.J. Salzborn for the help and support he gave me as my supervisor. Further, I would like to thank Professor R.B. Potts for his assistance and supervision when Dr. Salzborn was on study leave.

My thanks are also extended to Mrs. A. Walker, formerly of the Bureau of Transport Economics, for suggesting the problem which culminated in this thesis and also to Dr. D. Williams, the General Manager of the Australian National Railways, and his staff who provided assistance and invaluable discussions.

The typing of the thesis was ably performed by Ms. E. Henderson to whom I am greatly indebted.

Finally, I thank my wife Janet who stood by me and assisted me throughout the duration of my candidature.

CHAPTER 1

INTRODUCTION

There is a vast literature on the scheduling of job-shop processes. The first attempt to unify this literature was by Conway et al. [6] in 1967 and since then two other notable books, those of Baker [2] in 1974 and Rinnooy Kan [30] in 1976, have extended and formalized much of this material. More recently, papers like that of Graham et al. [12] have surveyed the literature and presented an up to date summary on the state-of-the-art for certain problem areas. An area which has received very little attention is the problem of scheduling a job-shop process in which the facilities for storing partially completed jobs have limited capacity. This is the topic of the thesis.

Such is the paucity of the literature on problems with restricted intermediate storage that one can detail the important contributions separately. First, there is a small group of papers which consider the so-called "no-wait" problems, in which, once a job starts on its first machine, it has to be processed by all its machines without stopping. This is equivalent to having no intermediate storage facilities. These papers are almost exclusively concerned with minimizing the time to complete all jobs (i.e. makespan) for a flowshop process, the main result being that this problem can be shown to be equivalent to a Travelling Salesman problem (see, for example, [13], [28] and [33]). Two exceptions are the papers by Van Deman and Baker [32] and Reddi and Ramamoorthy [29], the former being concerned with minimizing mean completion time, whilst the

latter is involved with applying branch and bound and longest path techniques to a more general process.

The other relevant contribution to the literature is the paper by Dutta and Cunningham [9] which proposes a Dynamic Programming procedure to solve a special two-machine, flowshop process with an intermediate storage facility of finite capacity. This and the above "no-wait" papers are so specialized that they do not provide any insight into the intricacies of the general problem. The purpose of this thesis is to expose these intricacies and to develop a basic framework for solving any job-shop problem for which there is restricted intermediate storage. The need for such a framework should be self-evident, since it will enable many realistic processes to be modelled more accurately.

There are a number of interesting differences between the restricted storage process and the more usual job-shop process with unrestricted intermediate storage. Firstly, when a partially completed job is waiting to be processed by its next machine, a distinction now has to be made between including or excluding the endpoints of the interval representing the waiting time. The alternatives produce the substantially different models developed in Chapter 2. Another difference concerns the definition of a *semi-active schedule* [6; p. 109]. In the thesis, schedules which concur with the basic philosophy of this definition are in fact defined as being semi-active; but, in contrast to the usual case, they do not possess the property that there is exactly one of them per feasible ordering of the jobs on the machines. This means that it is no longer sufficient to have solution procedures which just consider those orderings of the job components which produce *different*

processing orders on the machines. Instead, many more orderings of the job components may now need to be considered. The culmination of Chapter 2 is the presentation of general branch and bound procedures which take this fact into account.

In Chapters 3 and 4, two specializations of the general process are discussed separately. First, a flowshop process for which there is a storage facility at the end of each machine is considered for the two situations with *closed* (i.e. both endpoints included) and *half-open* (i.e. only one endpoint included) waiting intervals respectively. Next, in Chapter 4, a situation which can be interpreted as a train scheduling problem is investigated, allowing only for closed waiting intervals. Here the trains take the role of jobs, the sections of track correspond to machines and the stations take the place of the storage facilities. In each of the above cases a branch and bound solution procedure which avoids duplication of schedules is developed.

The single-machine scheduling problems with non-simultaneous release times usually have to be solved by implicit enumeration techniques. In Chapter 5, methods for finding lower bounds for some of these problems are presented and discussed. Most of these methods are based on known procedures for solving problems with simultaneous release times or other special properties. Applying many of them to the problem of determining lower bounds for the more general situations with non-simultaneous release times is believed to be an innovation in this area. The bounds thus obtained are used to derive lower bounds on all completions of a partial schedule for the general process described in Chapter 2. Such lower bounding methods could then be included in the branch and bound

solution procedures.

In the concluding chapter a number of areas are highlighted where further research could be undertaken.

CHAPTER 2

THE GENERAL PROBLEM

In the most general terms the problem which is addressed here is that of scheduling a job-shop process with the additional restriction that the capacities of the storage facilities for holding partially completed jobs are limited. This chapter is primarily concerned with describing the general problem and highlighting the complications which arise compared with the more usual situation where storage facilities are unrestricted. Further, some concepts, terminology and assumptions which are required here and in subsequent chapters are presented together with their raison d'être.

2.1 PROBLEM DESCRIPTION

2.1.1 Notation

Consider a job-shop consisting of m machines on which the components of n jobs are to be processed. Around the shop there are q storage locations, henceforth called *bins*, where partially completed jobs are assigned to wait until they can be processed by their next machine. Although one might expect a storage facility to be associated with each machine, this is not a necessary requirement and will not be assumed here. The bins are labelled $0,1,\ldots,q,q+1$, where 0 and $q+1$ denote the locations for unstarted and completed jobs respectively.

In keeping with the literature, it is convenient to define

operation O_{ij} as being that component of job j which is processed by machine i. (2.1.1)

This terminology assumes, as we shall here, that a job is processed by each machine exactly once.

The following data is required for each job $j(j=1,\ldots,n)$, machine $i(i=1,\ldots,m)$ and bin $\ell(\ell=1,\ldots,q)$:

Z_ℓ - the number of jobs which can be stored concurrently in bin ℓ; (2.1.2)

r_j - the earliest starting time of job j; (2.1.3)

d_{ij} - the duration of operation O_{ij}. (2.1.4)

For convenience, Z_ℓ will henceforth be called the *capacity* of bin ℓ. Further it is assumed that there are no restrictions on the number of jobs which can be present in locations 0 and q+1. This is expressed by setting

$$Z_0 = Z_{q+1} = n.$$

Clearly this is a desirable situation in most applications, since, in reality, jobs do not usually need to arrive at the job-shop until they are to be started and can leave the shop as soon as they are completed.

In some situations it may also be necessary to know a *due-date* or a priority *weighting* for job $j(j=1,\ldots,n)$. These quantities are denoted by d_j and ω_j respectively. One further item of data which can be included in some applications is a quantity, $Sh_{j\ell}$ say, which corresponds to the minimum time job j has to wait in bin ℓ. This information is rarely included in a job-shop process, but in Chapter 4 its presence enables a train scheduling situation to be modelled more accurately. Finally, as is usual for most scheduling problems, the data is assumed to be given in whole

values; in other words, all quantities are assumed to be *integers*.

To completely specify the path a job takes through the job-shop, it is necessary to know the order in which its operations are performed and also its movement into and out of bins. This can be accomplished by adopting the notation,

$p_{jk} = i$ if for job j, operation O_{ij} directly follows operation O_{kj}; (2.1.5)

and

$g_{ij} = \ell$ if job j goes directly to bin ℓ after operation O_{ij} has been completed. (2.1.6)

The entries of the $n \times m$ array

$$P = (p_{jk})$$

are machine numbers, whilst the $m \times n$ array,

$$G = (g_{ij}) ,$$

has bin numbers for its elements. Both matrices P and G are assumed to be part of the information which is available before the scheduling process is begun.

An entry,

$$p_{jk} = m+1 ,$$

is used to denote that operation O_{kj} is the last one to be performed before job j is completed. Similarly, if O_{ij} is the last operation to be performed for job j, then

$$g_{ij} = q+1$$

indicates that the completed job leaves the shop. Thus, the value $m+1$ appears exactly once in each row of P whilst $q+1$ appears exactly once in each of the n columns of G. Further, since

entries in G correspond to the bins which a job *enters*, the value 0 does not appear.

Example 2.1.1

As an example of how matrices P and G give full information on the movements of jobs, consider a situation with 2 jobs, 3 machines and 2 bins for which,

$$P = \begin{pmatrix} 2 & 3 & 4 \\ 2 & 4 & 1 \end{pmatrix}$$

and

$$G = \begin{pmatrix} 1 & 1 \\ 2 & 3 \\ 3 & 2 \end{pmatrix}$$

First, recall that the fact that the entries p_{13} and p_{22} both equal 4 indicates that O_{31} and O_{22} are the last operations to be performed for their respective jobs. Then, from the first row of P it can be seen that job 1 visits the machines in the order 1,2,3. But g_{11} equals 1 and therefore job 1 waits in bin 1 between the execution of operations O_{11} and O_{21}. Similarly, since g_{21} equals 2, job 1 enters bin 2 after the execution of O_{21} and therefore the path of job 1 is:

bin 0, machine 1, bin 1, machine 2, bin 2, machine 3, bin 3.

Likewise, the path for job 2 can be ascertained as being:

bin 0, machine 3, bin 2, machine 1, bin 1, machine 2, bin 3.

The unique path for each job is summarized schematically in figure 2.1.1. For clarity, the dummy bins 0 and 3 have been omitted.

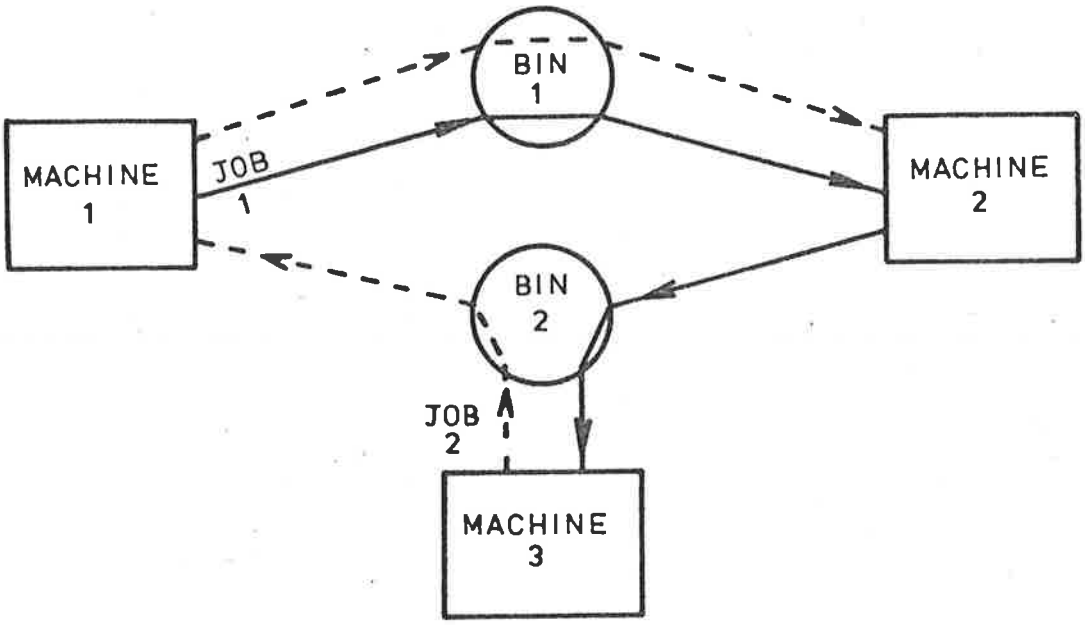


Figure 2.1.1

Special Structures

The two situations to be investigated in subsequent chapters of the thesis are special cases of the general model due to the specialized structure of the corresponding P and G matrices. In the first situation, the so-called flowshop process, each job follows the same path through the shop and therefore the machines are indexed so that for each j (j=1,...,n) ,

$$p_{jk} = k+1 \ . \qquad (2.1.7)$$

It is also assumed that there is a bin at the end of each machine into which each job goes when it is finished on that machine. Thus,

$$q + 1 = m$$

and for each operation O_{ij} $(i=1,\ldots,m;\ j=1,\ldots,n)$,

$$g_{ij} = i \quad . \tag{2.1.8}$$

The second situation can be interpreted as a train scheduling process where trains travel in one of two directions along a single-track railway line. The sections of track correspond to machines and are indexed so that one type of train, the "up-train", travels along the sections in the order $1,2,\ldots,m$, while the other type of train, the "down-train", travels along them in the order $m,m-1,\ldots,1$. Therefore,

$$p_{jk} = \begin{cases} k+1 & \text{if } j \text{ is an up-train;} \\ k-1 & \text{if } j \text{ is a down-train.} \end{cases} \tag{2.1.9}$$

For this situation the bins are actually the stations between the sections of track and therefore, if bin ℓ represents the station between sections ℓ and $\ell+1$, then

$$q + 1 = m$$

and

$$g_{ij} = \begin{cases} i & \text{if } j \text{ is an up-train;} \\ i-1 & \text{if } j \text{ is a down-train.} \end{cases} \tag{2.1.10}$$

To avoid separately specifying the entries in P and G corresponding to the *last* operation to be performed for each down-train, the usual $m+1$ and $q+1$ have both been replaced by 0. This causes no loss of generality since all these values are dummy quantities and 0

does not normally appear in the P and G matrices anyway.

2.1.2 Feasibility Conditions

To specify a particular schedule uniquely, it is sufficient to know a start time for each operation. Thus for operation O_{ij}, denote its *start time* by a_{ij} and for convenience its finish time, $a_{ij} + d_{ij}$, can be designated b_{ij}. For any schedule satisfying the technological requirements that the components of a job are to be processed in a prescribed order, these values a_{ij}, b_{ij} have to satisfy the constraints:

$$a_{ij} \geqslant r_j \quad \text{if } O_{ij} \text{ is the first operation for job } j; \tag{2.1.11}$$

and

$$a_{ij} \geqslant b_{kj} \quad \text{otherwise, where } p_{jk} = i \,. \tag{2.1.12}$$

Further, considering all jobs together, the following additional constraints must be satisfied: *At any time,*

1. There is no more than one job being processed by each of the machines; and (2.1.13)
2. The waiting intervals of not more than Z_ℓ jobs overlap at bin ℓ. (2.1.14)

This first constraint is the usual one for job-shops. The latter constraint is the one which sets this problem apart from the usual situation. The conditions (2.1.11) to (2.1.14) will henceforth be called the *feasibility conditions*.

2.1.3 Interpretations of Waiting Times

At this stage it is appropriate to discuss two interpretations of when a job is *actually* waiting in a bin. The first interpretation is to say that job j waits in bin ℓ during the time interval

$[b_{kj}, a_{ij}]$ where $g_{kj} = \ell$ and $p_{jk} = i$. The square brackets indicate that the endpoints b_{kj}, a_{ij} are included in the waiting time. Clearly a job may be allocated to the same bin more than once before it is completed and so there may be a number of disjoint time intervals corresponding to job j waiting in bin ℓ. Since the start and finish times of operations are included as waiting times, job j is considered as being processed by machine i during the time interval (a_{ij}, b_{ij}) *not* including the endpoints. Thus, even a job which does not actually *stop* in its assigned bin is modelled as though it waited there for an instant as it passed through. This interpretation will henceforth be referred to as the one for which the *waiting intervals are closed.*

The alternative interpretation is to allow some jobs to avoid passing through some of their assigned storage bins. This can be accomplished only when a job can begin its processing on its next machine as soon as it finishes on the current machine. Mathematically, this situation can be modelled in two ways: One is to use half-open intervals $(b_{kj}, a_{ij}]$ or $[b_{kj}, a_{ij})$ where $p_{jk} = i$ to indicate a time for which job j stays in bin g_{kj}. Corresponding to this representation, job j is processed by machine i during the time interval $(a_{ij}, b_{ij}]$ or $[a_{ij}, b_{ij})$ respectively. The other representation is to use the open interval (b_{kj}, a_{ij}) to specify the time job j spends in bin g_{kj} between the execution of the adjacent operations O_{kj} and O_{ij}. For any of these representations, if b_{kj} equals a_{ij}, then the waiting interval is empty and this is equivalent to job j *bypassing* bin g_{kj} and moving directly to machine i as soon as it finishes on machine k. Further, giving jobs the opportunity to bypass bins is one reason why the quantities

$Sh_{j\ell}$ are not included in the general model; that is, if a job has to wait for a specified minimum time in each of its assigned bins, then only when $Sh_{j\ell}$ is zero would job j be allowed to bypass bin ℓ. Finally, these models in which jobs can bypass bins will be referred to collectively as those for which the *waiting intervals are not closed.*

Example 2.1.2

To appreciate the difference between the use of closed and non-closed waiting intervals, consider the process represented earlier by figure 2.1.1. If the capacities of bin 1 and bin 2 are both 1, then the schedule represented by figure 2.1.2 would be feasible when waiting intervals are not closed but infeasible otherwise. This follows since, for the former situation, job 2 bypasses bin 1 and does not contribute to its capacity.

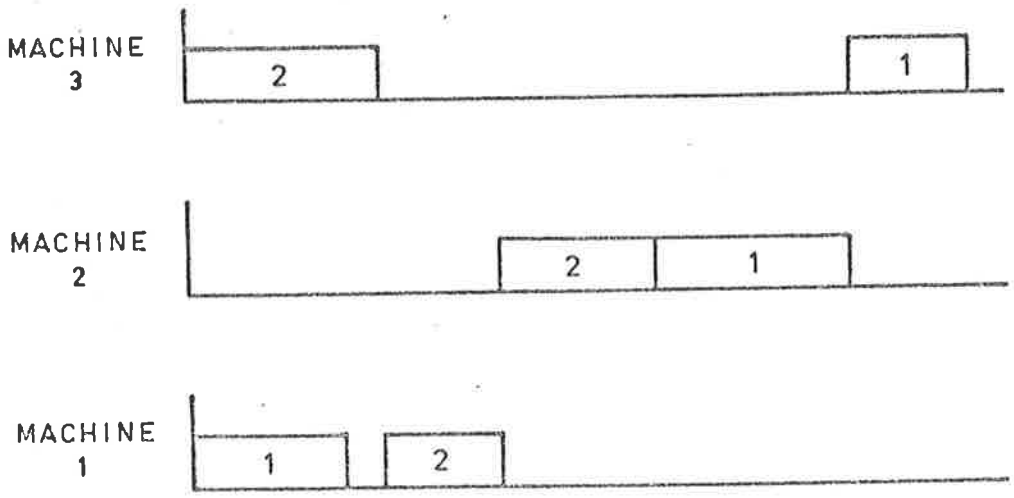


Figure 2.1.2

2.1.4 Optimality Criteria

An operation O_{ij} is said to be *scheduled* if a finite time interval $\{a_{ij}, b_{ij}\}$ is assigned to it; where, depending on the

interpretation described above, the interval is open, half-closed or closed. The braces are used to specify that exactly one of these situations is allowed. A *complete feasible schedule* is a complete set of scheduled operations satisfying the feasibility conditions (2.1.11) to (2.1.14). Given a complete feasible schedule, the *completion time* of job j is the value b_{ij} associated with operation O_{ij}, the last one to be executed for job j. For convenience, denote this completion time by c_j.

Throughout the thesis the usual practice of considering criteria which are non-decreasing functions of the completion times is adopted. Such criteria are termed *regular measures* ([6], [30]) and are represented by real functions, $f(c_1,\ldots,c_n)$, such that

$$f(c_1,\ldots,c_n) < f(c_1',\ldots,c_n') \text{ only if } c_j \leq c_j' \text{ for at least one } j. \qquad (2.1.15)$$

As indicated by Rinnooy Kan ([30], p.16), this class of regular measures

> "... is sufficiently rich to accommodate a large variety of criteria."

Also, considering such measures enables attention to be restricted to a finite subset of the infinite number of possible schedules associated with any problem instance. The aim therefore is to find a complete feasible schedule for which $f(c_1,\ldots,c_n)$ is minimal.

Some of the criteria which are considered in this thesis are:

1. *total completion time:* $f(c_1,\ldots,c_n) = \sum_{j=1}^{n} c_j$;
2. *total weighted completion time:* $f(c_1,\ldots,c_n) = \sum_{j=1}^{n} w_j c_j$;

3. *the number of tardy jobs:* $f(c_1,\ldots,c_n) = |\{j:c_j > d_j\}|$;

4. *the maximum lateness:* $f(c_1,\ldots,c_n) = \max_{1\leq j\leq n} (c_j - d_j)$;

5. *the maximum completion time:* $f(c_1,\ldots,c_n) = \max_{1\leq j\leq n} c_j$;

but it should be stressed that most of the results are general and applicable to all regular measures. The reasons for studying these particular criteria are twofold: Firstly, some are traditional measures which have had a great deal of literature devoted to them already (for example, criteria 1,4 and 5) and secondly, they occur as criteria for particular applications.

Symbolically, criteria 4 and 5 are represented by L_{max} and C_{max} respectively; while criterion 3 is represented by $\sum_{j=1}^{n} U_j$, where for a complete feasible schedule,

$$U_j = \begin{cases} 1 & \text{if } c_j > d_j \ ; \\ 0 & \text{otherwise} \ . \end{cases} \qquad (2.1.16)$$

Further, for any particular problem instance, since the earliest starting time r_j plus the total processing time $\sum_{i=1}^{m} d_{ij}$ is a known constant for each job j, then minimizing $\sum_{j=1}^{n} w_j c_j$ (and therefore $\sum_{j=1}^{n} c_j$) is equivalent to minimizing the total weighted *delay* incurred by the jobs.

2.2 SOLUTION SCHEME FOR CLOSED WAITING INTERVALS

To avoid confusion it is convenient to first concentrate just on the case where waiting intervals are closed, rather than trying to discuss all situations simultaneously. Before describing the

solution scheme for this situation, it is necessary to first resolve a problem of interpretation and then to provide some additional terminology.

2.2.1 Interacting Intervals

The problem to be resolved is that of deciding when two waiting intervals *interact* (as opposed to overlap) with each other. Since the time-intervals are closed it is possible, in theory, for two intervals to be separated by an infinitesimally small amount without overlapping; but the instant that the intervals "touch" (i.e. have an endpoint in common), they then overlap. Diagramatically this situation is represented by figure 2.2.1. Here the two intervals $[a,b]$ and $[b+\varepsilon,c]$ do not overlap as long as ε is positive. When ε equals zero the two intervals then have an endpoint in common and are considered to be overlapping at this time point.

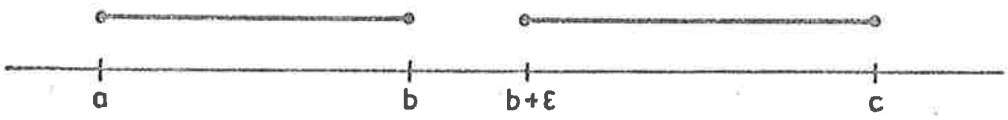


Figure 2.2.1

The reason that this situation is undesirable is because it is impossible to always characterize feasible schedules satisfactorily. For example in figure 2.2.1, if ε is positive and the two intervals correspond to two jobs waiting in a bin of capacity 1, then it is conceivable that the job which arrives at the bin at time $b+\varepsilon$ could

arrive there at any earlier time *greater* than b without effecting the feasibility of the schedule. It would be impossible in this case to decide which of the infinite number of almost identical schedules would be the most suitable one. The problem stems from the fact that the endpoint b of the interval $[a,b]$ is an *unattainable* lower limit for the endpoint $b+\varepsilon$ of the interval $[b+\varepsilon,c]$. If this lower limit were attainable, then clearly, with respect to regular measures, the schedule for which the job arrived in the bin at this lower limit would be the most suitable. Therefore, to overcome the problem that waiting intervals can get arbitrarily close to each other without overlapping, one can say that two intervals *interact* if the distance between them is less than a critical value δ (> 0), say; this then makes available an attainable lower limit for the endpoints. Thus, for the above example, only those schedules for which ε takes at least the value δ are feasible and, for regular measures, the schedule for which ε equals δ is the most suitable of these.

To include the idea of interacting intervals into the general model, it is necessary to refine the definition of a feasible schedule by altering feasibility condition (2.1.14) to:

> At any time, the number of jobs whose waiting intervals *interact* at bin ℓ is not more than the capacity, Z_ℓ. (2.2.1)

This condition assumes that the critical value, δ, has been prescribed beforehand. The difference between interacting intervals and overlapping intervals is of little consequence unless the number of jobs visiting a bin over some short period of time is greater than the capacity of the bin. In this case it may happen that the number of overlapping intervals is less than or equal to the capacity of the bin, whereas the number of interacting intervals is greater than this

capacity.

An interesting observation is that, given the critical value δ, two intervals $[a,b]$ and $[c,d]$ *interact* if and only if the open intervals $(a - \frac{\delta}{2}, b + \frac{\delta}{2})$ and $(c - \frac{\delta}{2}, d + \frac{\delta}{2})$ *overlap*. Further, since the data are integer values, by choosing δ equal to unity, one can restrict attention just to those schedules for which the values a_{ij} and b_{ij} $(i=1,\ldots,m;\ j=1,\ldots,n)$ are integers. In this case there would be no need to alter feasibility condition (2.1.14), since for such schedules, if two intervals interact, then they would have at least one integer time point in common and would therefore also overlap.

Restrictions on δ

Consider the case where a job j visits the same bin twice as in figure 2.2.2. Here, the two waiting intervals $[b_{i_1 j}, a_{i_2 j}]$ and $[b_{i_2 j}, a_{i_3 j}]$ *interact* whenever δ is greater than $d_{i_2 j}$. This situation is clearly unsatisfactory especially if the bin only has a capacity of 1.

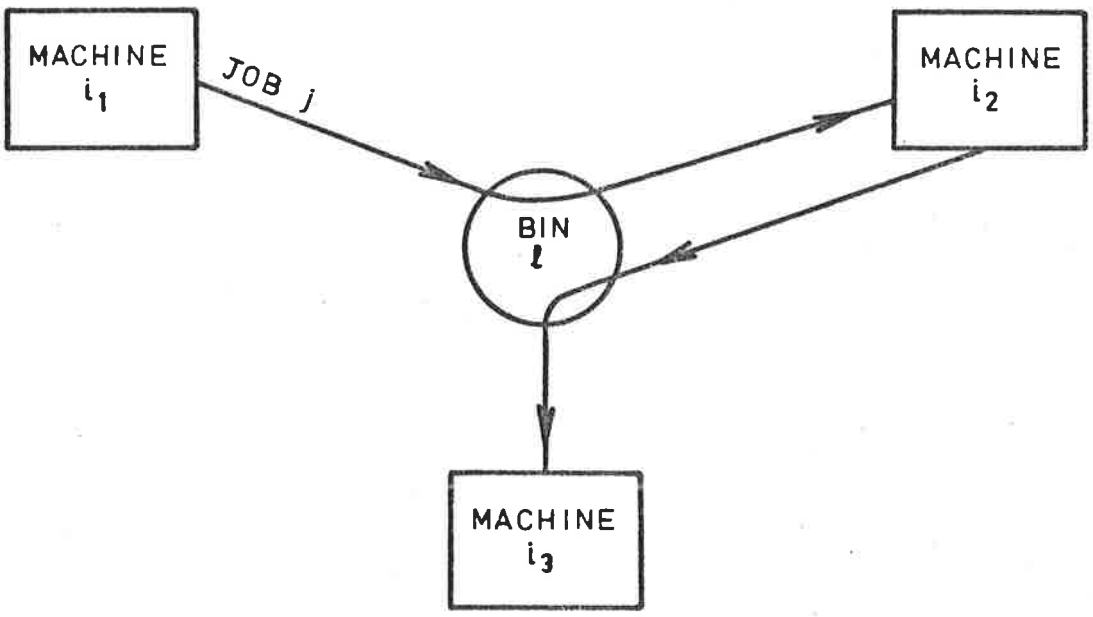


Figure 2.2.2

As another example of an unsatisfactory situation, consider a flowshop process in which job j_1 is processed by machine i *immediately* before j_2 (i.e. b_{ij_1} equals a_{ij_2}). Here, if δ is greater than d_{ij_2}, then the waiting intervals $[b_{ij_1}, a_{(i+1)j_1}]$ and $[b_{ij_2}, a_{(i+1)j_2}]$ interact since

$$b_{ij_2} = a_{ij_2} + d_{ij_2} < b_{ij_1} + \delta \leqslant a_{(i+1)j_1} + \delta \quad .$$

This interaction is independent of the value of $a_{(i+1)j_1}$ and, in fact, can only be avoided if job j_2 is not allowed to be processed by machine i until at least $\delta - d_{ij_2}$ units after O_{ij_1} is completed. This situation is undesirable since it causes unnecessary idleness of the machines.

In both the above examples, if

$$\delta \leqslant \min_{i,j} d_{ij} \quad ,$$

then neither situation can occur. Therefore, since the durations d_{ij} are all positive integers, it is sufficient to assume that henceforth δ is at most unity. This assumption does in fact enable the majority of unsatisfactory situations to be avoided.

2.2.2 Semi-Active Schedules

Now, since only regular measures are considered, the only schedules of interest are so-called semi-active schedules. Formally, for our purposes,

> a *semi-active schedule* is a complete feasible schedule for which the start time of no individual operation can alone be reduced without making the schedule infeasible or altering the order in which jobs are processed by some machine. (2.2.2)

This definition does not mean that there is a unique semi-active schedule corresponding to each technologically feasible ordering of the jobs on the machines. Instead, there may be a number of different feasible schedules which have the same ordering of the jobs on the machines, but for which, reducing the start time of any *single* operation would cause the number of interacting waiting intervals for some bin to exceed the capacity. Each such schedule is semi-active.

It is conceivable that the starting times of two or more operations could be reduced *simultaneously* to give a feasible schedule with the same ordering of the jobs on the machines, while, at the same time, reducing the start time of any individual operation would result in an infeasible schedule. Although the former schedule would be semi-active and clearly superior to the other semi-active schedules, it is convenient in the solution procedure to be able to consider operations separately. This explains why individual operations are specified in definition 2.2.2.

Example 2.2.1

As an example of how different semi-active schedules can correspond to the same ordering of the jobs on the machines, consider the problem with 3 jobs, 4 machines and 3 bins for which,

$$P = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 1 \\ 5 & 4 & 1 & 3 \end{pmatrix}$$

and

$$G = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{pmatrix} 2 & 4 & 2 \\ 4 & 1 & 3 \\ 3 & 2 & 2 \end{pmatrix}$$

The information from these G and P matrices is summarized by figure 2.2.3 (without dummy bins). For the data in Table 2.2.1

and specifying the critical value δ as being unity, the schedules represented by the Gantt charts in figure 2.2.4 have the jobs being scheduled on the machines in the same order and are in fact both semi-active. The waiting intervals for each job in each bin are indicated by the table beside each schedule.

The critical situation occurs in bin 1 where, for schedule (a), if the starting time of operation O_{11} is reduced, the waiting interval of job 1 interacts with that of job 2, making the schedule infeasible. Similarly, in schedule (b), if O_{32} is reduced, job 2 interacts with job 1 in bin 1, again causing infeasibility.

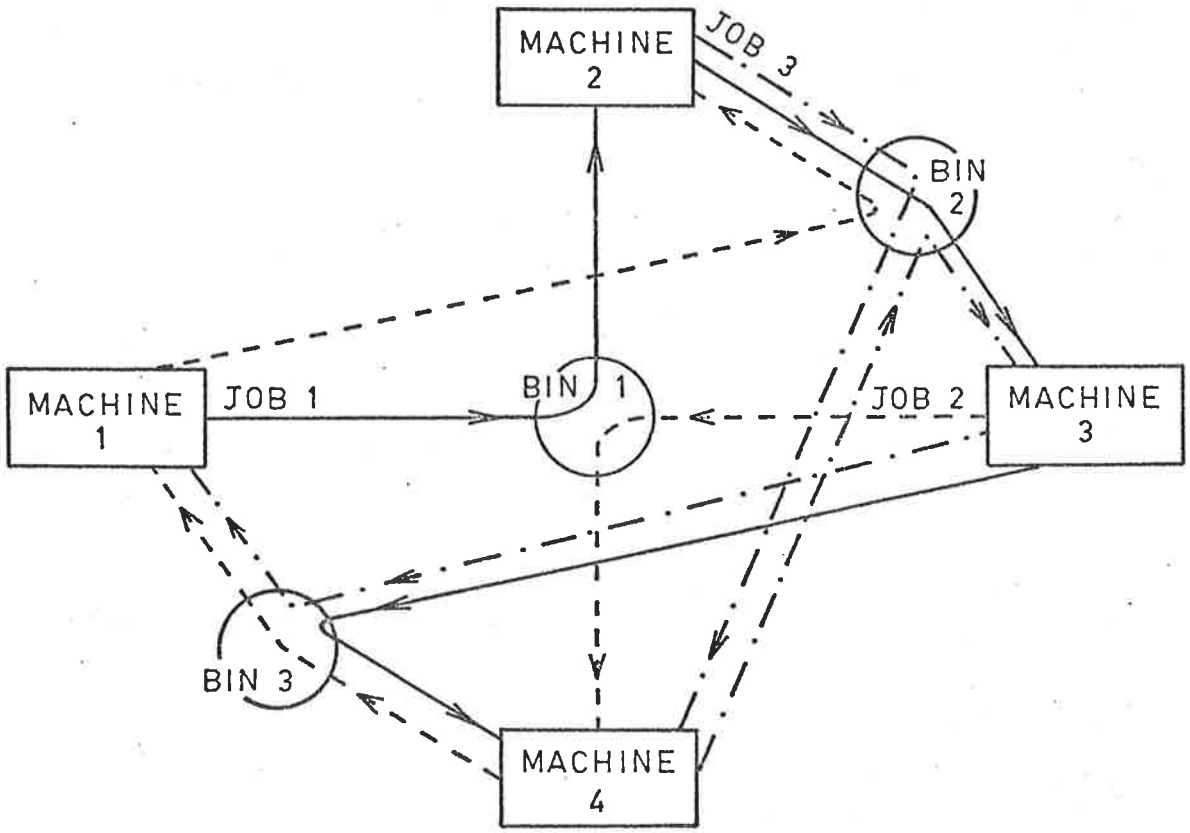


Figure 2.2.3

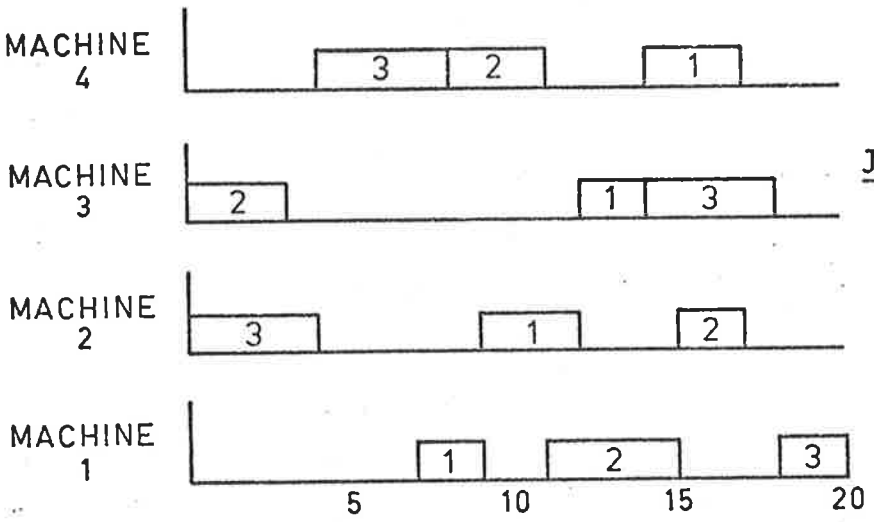
Job j	r_j	d_{1j}	d_{2j}	d_{3j}	d_{4j}
1	0	2	3	2	3
2	0	4	2	3	3
3	0	2	4	4	4

$Z_1 = 1;$

$Z_2 = 2;$

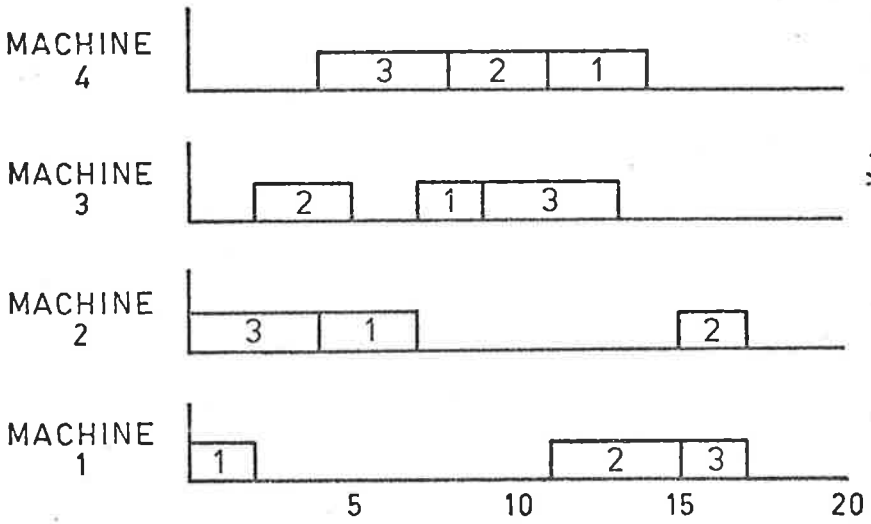
$Z_3 = 2.$

Table 2.2.1: Data for Example 2.2.1



JOB	BIN 1	BIN 2	BIN 3
1	[9,9]	[12,12]	[14,14]
2	[3,8]	[15,15]	[11,11]
3	—	[4,4],[8,14]	[18,18]

(a)



JOB	BIN 1	BIN 2	BIN 3
1	[2,4]	[7,7]	[9,11]
2	[5,8]	[15,15]	[11,11]
3	—	[4,4],[8,9]	[13,15]

(b)

Figure 2.2.4

As can be seen from this example, Gantt charts are probably not the best way of representing schedules, since it is difficult to see immediately what interactions occur in the bins. In subsequent chapters, the situations are sufficiently structured to enable space-time diagrams to be used for representing schedules. In these cases, the interactions in the bins are readily observed.

It is interesting to note that for the preceding example, even if definition (2.2.2) had been changed to allow for the start times of more than one operation to be reduced simultaneously, the schedules in figure 2.2.4 would *still* be semi-active.

Although there may be more than one semi-active schedule per technologically feasible ordering of the jobs on the machines, the total number of semi-active schedules for any problem instance is clearly still finite. Also, it is only for problems for which the bin capacities are small that feasibility condition (2.2.1) may dictate the start times of the operations. Therefore, for most orderings of the jobs on the machine, there will only be one corresponding semi-active schedule.

Further, if condition (2.1.14) had not been replaced by (2.2.1), the concept of semi-active schedule would be meaningless, since situations could occur where the endpoints of waiting intervals could be continually reduced to try and achieve unattainable lower limits. In such cases there would *always* be a better schedule available with respect to a regular measure.

2.2.3 The Solution Procedure

Now since, for semi-active schedules, each operation starts at an earliest time, then, for optimality criteria which are represented by non-decreasing functions of the job completion times (i.e., regular

measures), it is clear that at least one optimal schedule is semi-active. This explains why attention has been centred on semi-active schedules; that is, *in theory*, one could always obtain an optimal schedule by comparing the value of the criterion for each of the finite number of semi-active schedules and choosing the one with the smallest value. In practice though, this "finite number" is exceptionally large and very few instances of the general problem are amenable to this direct enumeration approach. Instead, implicit enumeration techniques have to be implemented.

A concept which is central to most of the algorithms presented in the thesis is that of *partial schedules* . The procedures are based on deciding how best to complete a current partial schedule. A set S of scheduled operations is called a *partial feasible schedule* (shortened to p.f.-schedule) if:

1. The feasibility conditions ((2.1.11)-(2.1.13) and (2.2.1)) are satisfied by the operations in S; (2.2.3)
2. $O_{ij} \in S$ and $i = p_{jk}$ implies $O_{kj} \in S$; (2.2.4)
3. $O_{kj} \in S$ and $O_{ij} \notin S$, where $i = p_{jk}$, implies $a_{ij} = \infty$; and $O_{ij} \notin S$, where i is the first machine on which job j is processed, implies $a_{ij} = \infty$. (2.2.5)

Thus, for a p.f.-schedule, each job is considered to wait indefinitely in the bin to which it was assigned to go after its last *scheduled* operation finished.

For a p.f.-schedule S, the *occupancy* of bin ℓ is defined by,

$$\theta_S(\ell) = \text{the number of jobs with } a_{ij} = \infty, \text{ where } i = p_{jk} \text{ and } \ell = g_{kj} . \quad (2.2.6)$$

If $\theta_S(\ell) < Z_\ell$, bin ℓ is said to be *unsaturated* in S. It is said to be *full* at any time for which the number of jobs whose waiting intervals concurrently interact is Z_ℓ. Bin ℓ being full does *not* necessarily mean that $\theta_S(\ell) = Z_\ell$, since some of the interacting intervals may correspond to jobs which, in S, have left the bin.

The basic *framework* for a procedure for generating semi-active schedules is now presented for the situation with closed waiting intervals. Some necessary notation is:

U = set of jobs that have not yet been completely scheduled; (2.2.7)

$K = \{k_j \mid j \in U\}$ where k_j is the machine on which the next operation of job j is to be performed; (2.2.8)

S = current schedule. (2.2.9)

The procedure is presented as a non-deterministic algorithm ([16]); that is, to generate all semi-active schedules one has to incorporate a suitable backtracking procedure.

Procedure S.A.1

Step 0: Set $U = \{1,\ldots,n\}$;

k_j = machine on which the first operation for job j is to be processed;

$a_{k_j j} = \infty$ for all $j \in U$;

$S = \phi$.

Step 1: Determine a set Q_S of operations, where

$O_{k_j j} \in Q_S$ if,

(i) $j \in U$, and

(ii) $\theta_S(\ell) < Z_\ell$ where $g_{k_j j} = \ell$.

Step 2: Restrict Q_S to a set of *schedulable* operations, Ω_S. If $\Omega_S = \phi$, stop.

Step 3: Select $O_{kj} \in \Omega_S$ and set $a_{ij} = \infty$, where $p_{jk} = i$. Calculate values a_{kj} and $b_{kj} = a_{kj} + d_{kj}$ such that

(i) operation O_{kj} is scheduled after all previously scheduled operations on machine k;

(ii) the feasibility conditions (2.1.11) to (2.1.13) and (2.2.1) are not violated.

(iii) a_{kj} is as small as possible.

Step 4: Add O_{kj} to S and update U and K.

Return to Step 1.

After each iteration of the steps of the procedure a p.f.-schedule S, say, is available. Furthermore, because of (iii) in Step 3, S is *semi-active in nature* since the start time of no individual operation in S can be reduced without making the schedule infeasible (Step 3(ii)) or altering the order in which the scheduled operations are executed on some machine (Step 3(i)).

2.2.4 Features of Procedure S.A.1

By choosing different *sequences* of operations from the sets Ω_S, it is possible to obtain two p.f.-schedules S and S' which contain the same operations, have the same orderings of the jobs on the machines, but which are not only different schedules but also semi-active in nature. This follows since bins may become *full* at different stages due to the different orders in which the operations have been chosen. An example of this phenomenon has

already been discussed (see example 2.2.1). In fact if, for this example, the operations are chosen from the sets Ω_s in the order,

$$O_{23}, O_{43}, O_{32}, O_{42}, O_{11}, O_{21}, O_{31}, O_{41}, O_{12}, O_{22}, O_{33}, O_{13},$$

then schedule (a) in figure 2.2.4 is obtained; whereas choosing the operations in the order,

$$O_{23}, O_{43}, O_{11}, O_{21}, O_{32}, O_{42}, O_{31}, O_{41}, O_{12}, O_{22}, O_{33}, O_{13},$$

results in schedule (b). Clearly though, it may be possible for a similar situation to occur for which no bins become full; in which case, different sequences of choices from Ω_s may result in the same semi-active schedule. In practice Ω_s is chosen in such a way that this duplication does not happen and in fact this explains why the set Q_s is restricted to Ω_s in Step 2. For each specialization of the general model, Q_s should be restricted to Ω_s in a way which guarantees that each generated semi-active schedule is obtained *uniquely*.

Further, for the general model it seems likely that every semi-active schedule can be generated by S.A.1, since it should be possible to decompose any semi-active schedule into a sequence of operations and then rebuild the schedule exactly using the procedure. The sequence provided by the decomposition of the schedule would ideally correspond to a sequence obtainable by choosing operations from the sets Ω_s. In this way, the decomposition algorithm would be the inverse procedure corresponding to S.A.1. In subsequent chapters, the specializations of procedure S.A.1 are shown to be capable of generating *every* semi-active schedule exactly once.

The ability to obtain different sequences of operations stems from the fact that, at each stage of the procedure, the choice from Ω_S is arbitrary. Indeed, it is this feature which makes procedure S.A.1 non-deterministic. Another point worthy of mention here is that, if at some stage the set Ω_S is empty, then the procedure terminates and the existing p.f.-schedule is termed *maximal*. In particular, all complete semi-active schedules generated by the procedure are maximal, since, if all operations have been scheduled, then Q_S and therefore Ω_S are empty.

Using Conway, Maxwell and Miller's terminology ([6], p.112), S.A.1 is a *single-pass procedure* since, once an operation has been assigned a starting time, this time is permanent and cannot be changed when a later operation is scheduled. In fact the procedure is a so-called *dispatching* procedure, which is a single-pass procedure for which starting times on any given machine are determined in such a way that they form a strictly non-decreasing sequence of values. This follows from Step 3(i) of the procedure.

2.2.5 Implementation of the Solution Procedure

The schedule generating procedure S.A.1 can be converted into a *backtrack algorithm* in the following way: At each stage, generate all the p.f.-schedules which can be obtained from the current schedule S by scheduling one of the operations in Ω_S. All but one of these descendant schedules are stored for later use. The procedure is then continued with the one which is retained. Whenever a maximal p.f.-schedule is obtained, the last stored p.f.-schedule is retrieved from storage and the process is continued with this one.

Further, if a routine for calculating a lower bound on the

values of *all* feasible completions of a p.f.-schedule is incorporated into the above backtrack procedure, then it is transformed into a branch and bound algorithm using the following rule:

> Whenever the lower bound associated with the current p.f.-schedule is larger than the value of a complete incumbent schedule, discard the current schedule and continue the procedure with a new one retrieved from storage.

Since lower bounds need to be calculated many times during the course of the algorithm, all computer programmes used to solve specializations of the general problem employ a very simple bounding routine, viz. determining a bound using *projected* job completion times. These times are obtained by assuming that, from the situation described by the current p.f.-schedule, all jobs are completed without incurring further delays. Clearly, these completion times, c_j say, are no larger than the actual completion times for *any* feasible completion of the current schedule. Hence, since for our purposes all optimizing criteria are regular measures, $f(c_1,\ldots,c_n)$ acts as a lower bound on the value of any such completion. Some work on deriving more sophisticated bounds for the different criteria is described in Chapter 5. In most cases the basic idea is to schedule operations on a single machine while ignoring the interactions between jobs on other machines. The value of the criterion thus obtained is used to provide a lower bound on the value for the whole problem.

In practice, to accelerate the branch and bound procedure slightly, it is sometimes advantageous to include a heuristic

procedure for quickly determining an initial incumbent schedule. In this way a realistic schedule and a corresponding value are available from the outset. This is beneficial to the solution procedure in that otherwise a number of incomplete maximal p.f. -schedules might be generated before an initial schedule is found; whereas, if a schedule is available from the beginning, then a lot of these dead ends might be avoided. Again, it is generally not worth expending a lot of time using a very sophisticated heuristic to obtain an initial incumbent schedule; experience indicates that a schedule which is generated quickly provides better overall results.

A further point on the implementation of a branch and bound procedure is that any of a number of different rules could be used to decide which descendant p.f.-schedule to keep at each stage and, when backtracking, which p.f.-schedule should be retrieved from storage. For example, one could choose the descendant schedule which is obtained by adding to the current schedule that schedulable operation (i.e. in Ω_S) which would finish first. Alternatively, one could choose the schedulable operation which could begin first. Other rules could be based on choosing the schedule which exhibits some feature which might reflect well on the value of the criterion. For the results quoted in this thesis, the descendant schedule chosen at each stage is the one for which the last scheduled operation was the one which began first amongst the schedulable operations. Finally, for actual implementation of the branch and bound procedure, it is convenient to continually *update* the starting times for those operations which have not yet been scheduled but whose predecessors have. Thus, suppose operation O_{kj} is scheduled and consider the unscheduled operation O_{ij}, where $p_{jk} = i$. After scheduling O_{kj},

the occupancy of storage location g_{kj} has increased by one, but instead of setting a_{ij} to infinity, it is set to tentatively take the value for which O_{ij} satisfies the three conditions of Step 3 in S.A.1. There are two benefits to be derived from this policy: First, to schedule a schedulable operation at any stage, it is only a matter of noting that the corresponding tentative value is now permanent. Secondly, and more importantly, knowledge of tentative start times enables a better bound to be obtained at each stage. This follows since the tentative times are *minimum* values for the final times associated with these operations in *any* completion of a current p.f.-schedule and therefore provide a much stronger bound than if they were not known.

Whenever an operation is scheduled it is not necessary to update all tentative times. Instead, only those times for operations which are directly effected by the scheduled operation need updating. One drawback of the updating procedure is that, when one is backtracking, it is not just a matter of unscheduling the last scheduled operation; instead, the tentative start times of all operations effected by the unscheduling of this operation have to be updated. This disadvantage is usually far outweighed by the previously discussed benefits.

A useful feature of branch and bound procedures is that the incumbent schedule is only ever superseded by better schedules. Whence, one heuristic solution to a problem is to stop the procedure after a specified time and use the current incumbent as the solution.

2.3 SOLUTION SCHEME FOR NON-CLOSED WAITING INTERVALS

2.3.1 Open Waiting Intervals

In the same way that a distinction between interacting and overlapping waiting intervals has to be made when such intervals are

closed, a similar distinction has to be made between interacting and overlapping *processing* intervals when they are closed; that is, when waiting intervals are open. Thus, if δ is a prescribed positive value, two processing intervals $[a_{ij_1}, b_{ij_1}]$ and $[a_{ij_2}, b_{ij_2}]$ corresponding to machine i, *interact* if

$$a_{ij_1} < a_{ij_2} < b_{ij_1} + \delta$$

or

$$a_{ij_2} < a_{ij_1} < b_{ij_2} + \delta \qquad (2.3.1)$$

Further, by analogy with the alteration of condition (2.1.14) to accommodate the idea of interacting intervals, feasibility condition (2.1.13) would have to be changed to:

> At any time, the intervals associated with jobs being processed by the same machine do not *interact*. (2.3.2)

This condition requires each machine to be idle for at least δ time units between operations. This situation is not satisfactory in practice and is certainly not usual for job-shop processes. As an illustration of the problem, consider the following example:

Example 2.3.1

For the schedules represented by the Gantt charts in figure 2.3.1, schedule (a) is the one which usually represents two jobs being processed successively by the same machine; but, this is infeasible since *both* jobs are being processed at time t. The best feasible alternative is schedule (b) which is not desirable. Henceforth, because of the unsatisfactory way in which machines are utilized, situations with closed processing intervals will not be considered. Consequently, the remainder of this section is devoted entirely to the case where waiting intervals (and therefore processing intervals) are half-open. One final interesting observation on this

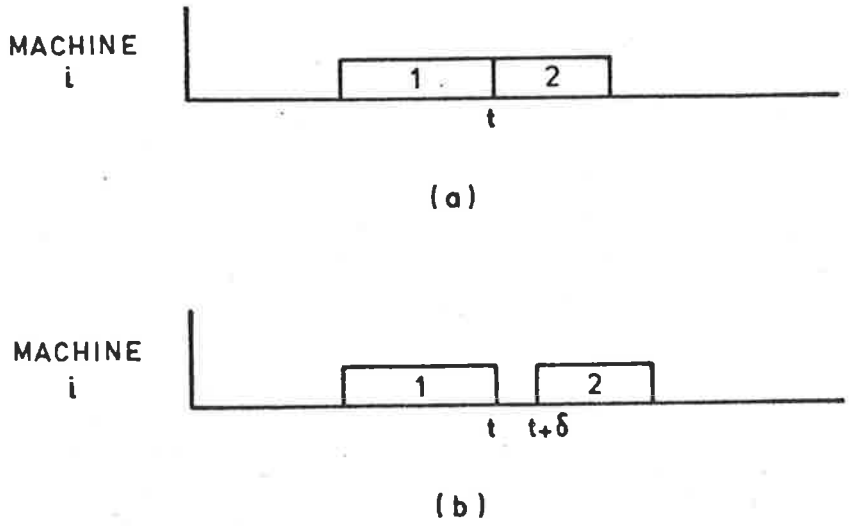


Figure 2.3.1

point is that feasibility condition (2.1.13) is common to most machine scheduling problems and so, although it is rarely, if ever, specifically stated, the usual situation (i.e. unrestricted intermediate storage) *assumes* that the processing intervals are open or only half-closed. In this way, the machines need not be idle when jobs are available.

2.3.2 Comparisons with Closed Waiting Intervals

When intervals are half-open, it is *unnecessary* to make a distinction between interacting and overlapping intervals, since it is now possible to reduce an endpoint to a value for which the schedule is feasible and so that any further reduction would result in it becoming infeasible; in other words, the lower limit for every endpoint is attainable. These comments apply equally well to waiting and processing intervals. Thus, the original feasibility conditions ((2.1.11) to (2.1.14)) can still be used to determine the feasibility of a schedule. Without loss of generality, waiting intervals and processing intervals can be represented by $[b_{kj}, a_{ij})$ and $[a_{ij}, b_{ij})$

respectively.

The No-Wait Problem

A good reason for studying the case where waiting intervals are half-open is to enable the so-called "no-wait" situations to be modelled accurately. Briefly, the adjective "no-wait" describes problems for which once a job starts on its first machine, it has to be processed by all its machines without stopping. Thus, the only waiting a job can do is in the dummy location 0. Clearly, the no-wait situation can be interpreted as a special case of the general model by requiring all but the dummy bins to have capacity zero and letting the waiting intervals be half-open. Further, it is not possible to achieve the same result by using closed waiting intervals and capacities of unity, since this alternative allows jobs to wait in bins for a machine to become idle.

The no-wait problems are usually restricted to the flowshop situation and the criterion of minimizing the maximum completion time (i.e. makespan). The main result to date is that such problems can be formulated as Travelling Salesman problems (see, for example, [13], [28] and [33]).

For any particular specialization of the general model, deciding whether to use closed or half-open waiting intervals is mainly dependent on the characteristics of the situation which one is attempting to model. For example, if jobs are to be allowed to bypass bins, half-open intervals must be used. On the other hand, the train scheduling problem described in Chapter 4 requires closed waiting intervals if a satisfactory model of reality is to be achieved. The schedules obtained for the two alternative types of

waiting intervals may be different for similar circumstances since, in the case of closed intervals, a separation of δ time units may possibly be required between a pair of intervals when the corresponding bin is full.

Example 2.3.2

Consider a 3 job, 2 machine flowshop process with a single storage bin of capacity one. The schedules (a) and (b) in figure 2.3.2 correspond to the closed and half-open interval situations respectively. In (a), job 3 can not enter the bin until δ time units after job 2 leaves there, whereas in (b), job 3 can enter the bin as soon as job 2 leaves.

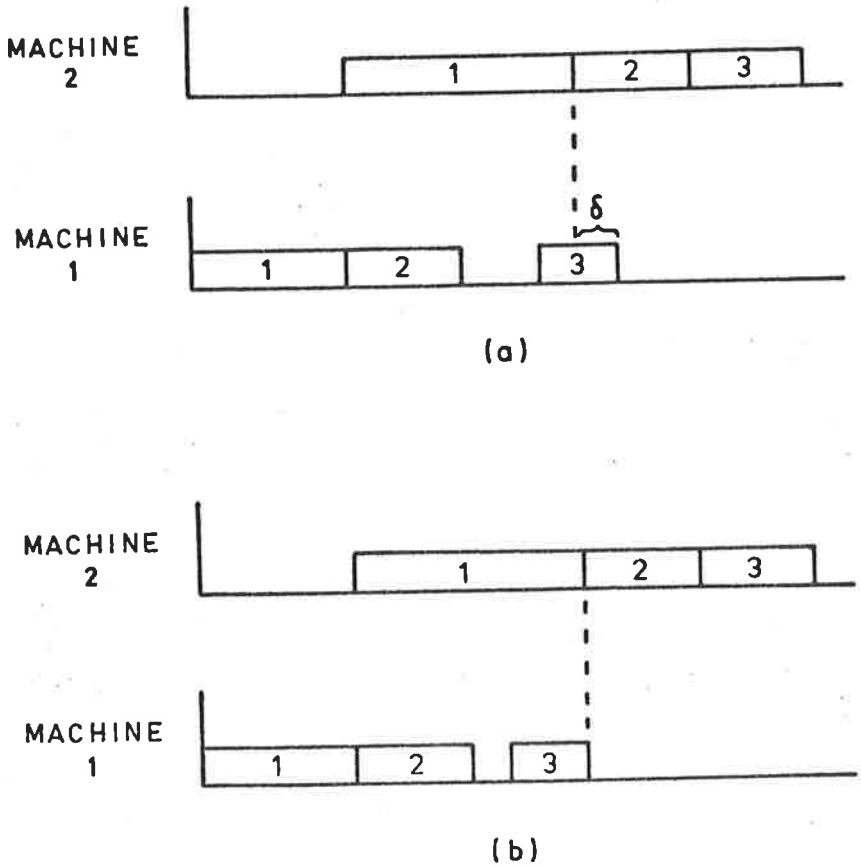


Figure 2.3.2

Further, due to the possibility of jobs bypassing bins, some schedules are permitted for situations with half-closed waiting intervals which are not allowed when the waiting intervals are closed. This is illustrated by the following example.

Example 2.3.3

Consider a simple 2 job, 2 machine flowshop process with a single bin of capacity one. For this situation, the schedule shown in figure 2.3.3 is *not* feasible if waiting intervals are closed, but is feasible when such intervals are half-open, since job 2 bypasses the full bin.

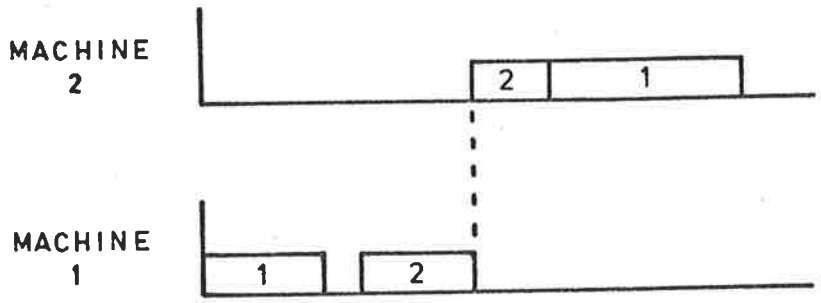


Figure 2.3.3

2.3.3 Semi-Active Schedules

The last example can be used to illustrate a significant difference between the theory for closed waiting intervals and that of half-open intervals. For, using the definition of semi-active schedule presented in the previous section and assuming intervals are half-open, the schedule in figure 2.3.3 would be semi-active,

since no *individual* operation can have its start time reduced without either causing infeasibility or changing the order in which jobs are processed on some machine. In particular, if the start time of operation O_{12} is reduced, then the waiting interval $[b_{12},a_{22})$ is no longer empty and therefore both jobs 1 and 2 would be waiting concurrently in the bin, the resulting schedule thus being infeasible. But, if the start times of operations O_{12} and O_{22} are reduced simultaneously by the same amount, then job 2 would still bypass the bin and, reducing the start time of operation O_{21} appropriately, the resulting schedule, as shown in figure 2.3.4, would still be feasible. Clearly, for any regular measure, this latter schedule is superior to the first.

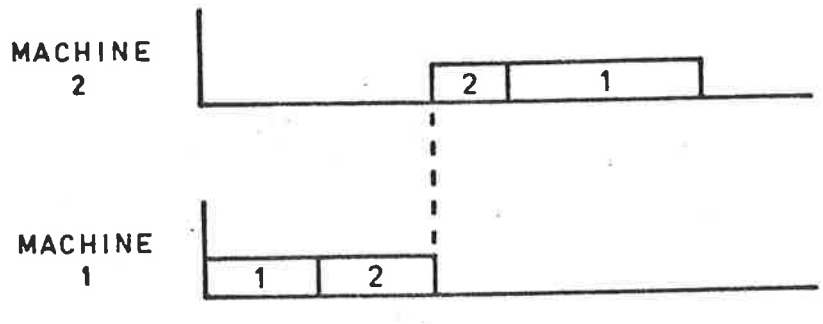


Figure 2.3.4

Further, there are an *infinite* number of feasible schedules between the extremes of figures 2.3.3 and 2.3.4, each of which corresponds to the same ordering on the machines and is semi-active by definition (2.2.2). Therefore, for the situation with half-open waiting intervals, the definition of semi-active schedules must be

altered to allow starting times to be reduced simultaneously. In this way only the schedule in figure 2.3.4 would be semi-active.

Now, for closed waiting intervals, the notion of reducing start times simultaneously was suggested and then discarded. The difference between that situation and the one discussed here is that, for half-open intervals, if a job bypasses a bin, then separately reducing the start time of the appropriate operation causes this job to no longer bypass the bin, a fundamental change to the nature of the schedule. On the other hand, with closed waiting intervals, jobs cannot bypass bins and such fundamental changes do not occur. For example, if for the 2 job, 2 machine flowshop process discussed above, the capacity of the bin is changed from 1 to 2 then the schedule in figure 2.3.3 is now feasible for closed intervals. But, it is not semi-active, since the start time of operation O_{21} can be reduced separately without making the schedule infeasible. Also, the start times of operation O_{22} and O_{21} can be reduced separately until, finally, the schedule in figure 2.3.4 is obtained. This schedule is the only semi-active schedule corresponding to the indicated ordering of the jobs on the machines.

In the same way that schedule 2.3.4 was obtained from 2.3.3 by allowing the start times of two operations to be reduced simultaneously, a similar situation can obviously occur on a larger scale when a job bypasses a number of full bins. In this case, it may be possible to simultaneously reduce the start times of a number of adjacent operations without causing infeasibility or altering the order in which jobs are processed by some machine. To overcome the deficiency of definition (2.2.2), it is convenient to treat adjacent operations for a job as a single entity and to

replace the term "operation" by this entity. Thus,

> a *string of operations* is any non-empty collection of adjacent operations corresponding to the same job. (2.3.3)

The notation,

$$\ulcorner O_{i_1 j}, O_{i_2 j}, \ldots, O_{i_q j} \urcorner \quad \text{where} \quad p_{j i_s} = i_{s+1} \quad (s=1,\ldots,q-1)$$

is used to represent a string of operations. Replacing the term "operation" by "string of operations", the new definition of a *semi-active schedule* for situations with half-open waiting intervals is:

> a complete feasible schedule for which the start time of no individual *string of operations* can alone be reduced without making the schedule infeasible or altering the order in which jobs are processed by some machine. (2.3.4)

Since a single operation is also a string of operations, any schedule which is semi-active by this new definition is also semi-active by the old definition. Further, it is only when jobs bypass full bins, as in the earlier example, that the concept of "strings of operations" is important. This follows since, if a bin is not full, then a large string of operations may be divided into two and the start times of these adjacent strings can be separately reduced in succession without violating the capacity restriction of this bin and making the schedule infeasible. Similarly, if a job does not bypass a bin, then there is no need to simultaneously reduce the start times of the adjacent strings of operations for this job. Treating them separately does not change the fundamental nature of the schedule.

2.3.4 The Solution Procedure

The procedure for generating schedules for the situation with half-open waiting intervals is analogous to that for the situation with closed intervals; except, of course, to produce p.f.-schedules which are semi-active in nature, one has to replace the notion of single operations by "strings of operations". Other than this alteration, the remaining terminology and notation is unchanged. The procedure, which is again non-deterministic, is:

Procedure S.A.2

Step 0: Set $U = \{1,\ldots,n\}$;

k_j = machine on which the first operation for job j is to be processed;

$a_{k_j j} = \infty$ for all $j \in U$;

$S = \phi$.

Step 1: Determine a set Q_S of strings of operations for which $\ulcorner O_{k_j j}, O_{i_1 j}, \ldots, O_{i_q j} \urcorner \in Q_S$, where $p_{j k_j} = i_1$ and $p_{j i_s} = i_{s+1}$ $(s=1,\ldots,q-1)$, if

(i) $j \in U$, and

(ii) $\theta_S(\ell_1) = Z_{\ell_1}, \ldots, \theta_S(\ell_q) = Z_{\ell_q}$ and $\theta_S(\ell) < Z_\ell$ where $\ell_1 = g_{k_j j}$, $\ell_s = g_{i_{s-1} j}$ $(s=2,\ldots,q)$ and $\ell = g_{i_q j}$.

Step 2: Restrict Q_S to a set of *schedulable* strings of operations, Ω_S. If $\Omega_S = \phi$, stop.

Step 3: Select a string $\ulcorner O_{i_1 j}, \ldots, O_{i_q j} \urcorner \in \Omega_S$ where $p_{j i_s} = i_{s+1}$ $(s=1,\ldots,q-1)$; and set $a_{ij} = \infty$ where $p_{j i_q} = i$. Calculate integers

$a_{i_1 j}, a_{i_2 j} = a_{i_1 j} + d_{i_1 j}, \ldots, a_{i_q j} = a_{i_{q-1} j} + d_{i_{q-1} j}$

and $b_{i_q j} = a_{i_q j} + d_{i_q j}$ such that

(i) for each s $(s=1,\ldots,q)$, operation $O_{i_s j}$ is scheduled after all previously scheduled operations on machine i_s ;

(ii) the feasibility conditions (2.1.11) to (2.1.14) are not violated;

(iii) $a_{i_1 j}$ is as small as possible.

Step 4: Add the operations $O_{i_1 j},\ldots,O_{i_q j}$ to S and update U and K. Return to Step 1.

The strings of operations in Ω_S are chosen so that, if they were scheduled next, all full bins would be bypassed. Clearly, as for the situation with closed intervals, a p.f.-schedule is available after each iteration of the steps of the procedure and such schedules are semi-active in nature. Further, by choosing different sequences of strings of operations from the set Ω_S, it is possible to obtain two p.f.-schedules which contain the same operations, have the same orderings of the jobs on the machines but which are actually different semi-active schedules. As for the case with closed waiting intervals, this situation occurs when the different sequences of strings of operations cause the bins to become full at different stages. Finally, Ω_S is normally chosen to avoid duplication of schedules, but not to eliminate schedules completely. As before, whenever Ω_S is empty, the procedure terminates and the existing p.f.-schedule is termed *maximal*.

In one sense procedure S.A.2 is a single pass procedure since, once a string of operations has been assigned a start time, this time is permanent and cannot be changed when a later string is scheduled. On the other hand, the situation can occur where a

start time cannot be assigned to an operation in a string from Ω_S until the start times of subsequent operations in its string have had their start times determined. But, since the operations in a string from Ω_S are scheduled *consecutively*, it would seem that S.A.2 is, for all intents and purposes, a single-pass procedure. If this dilemma is resolved as suggested, then S.A.2 would also satisfy the conditions for a *dispatching* procedure.

Finally, without spelling out the details, procedure S.A.2 can be incorporated into a branch and bound procedure in an analogous way to S.A.1. The only difference being that, whenever the term "operation" was used, it should now be replaced by "string of operations". For actual implementation, the book-keeping involved in dealing with strings of operations containing different numbers of operations is fairly involved and is certainly much harder than when only single operations have to be considered.

2.4 COMPARISONS WITH THE USUAL JOB-SHOP PROCESS

Although many of the ideas and concepts involved in the problems with capacitated storage bins are similar to those involved in the usual job-shop process with bins of unlimited capacity, there are a number of differences which should be highlighted.

2.4.1 Semi-Active Schedules

One of the major differences between the two processes is associated with the notion of semi-active schedules. Although the basic idea that operations start as early as possible is common to both circumstances, for the usual job-shop process it is only possible to obtain one semi-active schedule per technologically feasible ordering of the jobs on the machines. As has been seen

already, this is no longer the case when storage bins are capacitated. Further,when the waiting intervals are half-open, the definition of semi-active schedule has to be changed to include strings of operations, a major alteration to the usual case.

2.4.2 Active Schedules

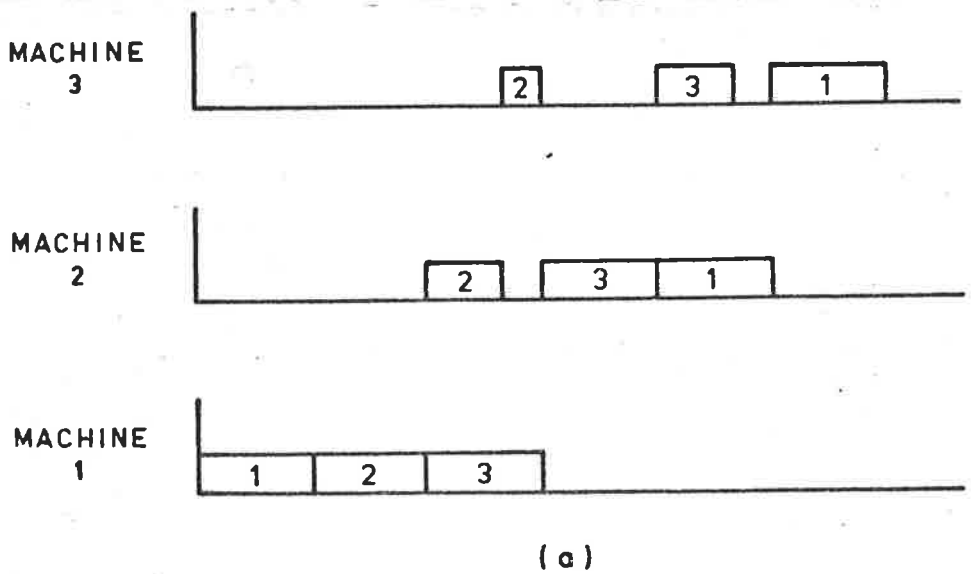
One feature of the usual job-shop process which has not yet been investigated in the context of the limited storage situation is the possibility of restricting attention to *active* schedules. In the literature the idea of an active schedule has been popular since Giffler and Thompson described it in 1960 ([11]). In essence, an active schedule is a feasible schedule for which it is impossible to decrease the starting time of any individual operation without producing an infeasible schedule or *increasing* the start time of at least one other operation. Thus, for active schedules, *every* operation starts as soon as it can and no machine is idle long enough to process an available job not already processed by it. For job-shop processes with unrestricted intermediate storage, at least one optimal schedule is active. Thus, since the set of active schedules is clearly a subset of the semi-active schedules, then the restriction to active schedules is certainly preferable.

Unfortunately, for the situation with restricted bin capacities, it is no longer true that it is optimal for every operation to commence at the *earliest* possible time as described above. This follows since it is conceivable that,by processing a job earlier on a machine,it may have to wait in a bin for a longer time than it did previously. This latter schedule may now no longer be feasible as illustrated by the following example:

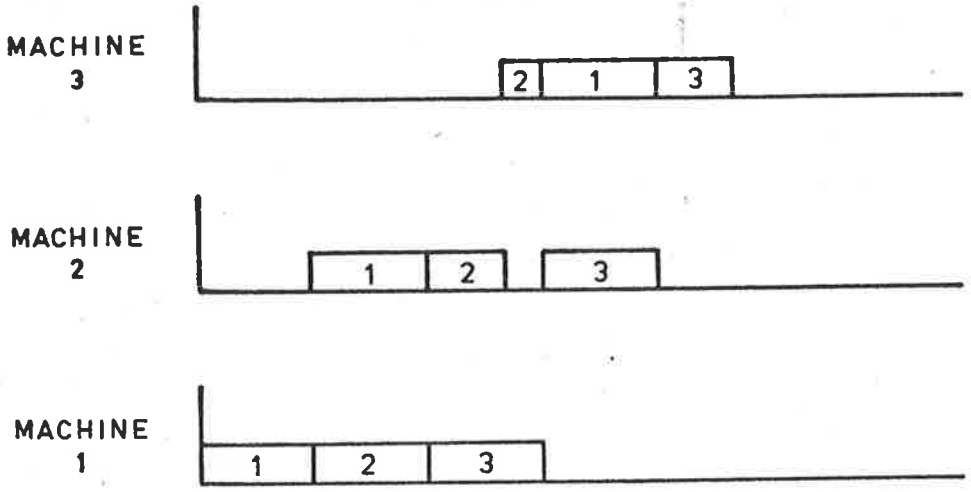
Example 2.4.1

Consider the schedules for the 3 job, 3 machine flowshop process depicted in figure 2.4.1. Here, suppose the bin between machines 1 and 2 has capacity 2 and the bin between machines 2 and 3 has capacity 1 and that the waiting intervals are closed. Then, schedule (a) is feasible, but not active, since machine 2 is idle long enough to process job 1 before job 2. On the other hand, schedule (b), which for the usual situation is the active schedule obtained from (a), is *not* feasible since the bin between machines 2 and 3 has two jobs interacting there concurrently, one more than its capacity.

To see that active schedules are again not suitable when waiting intervals are half-open, consider the above example again, but this time use half-open waiting intervals and bin capacities of 1 and 0 respectively.



(a)



(b)

Figure 2.4.1

A fact about active and semi-active schedules which is not particularly clear from the literature is that the number of possible different semi-active schedules is independent of the job data and is dependent only on the number of technologically feasible orderings. On the other hand, determining the number of different active schedules is problem specific and requires a complete knowledge of the data. For example, in figure 2.4.1, if operation O_{21} had been of longer duration than O_{12}, then schedule (a) would not only be feasible but also "active".

2.4.3 Concluding Remarks

When one is using a dispatching procedure to generate semi-active schedules for the usual job-shop process with unlimited intermediate storage, the start times of operations are determined just by the operations previously scheduled on the same machine and the scheduled operations for the same job. On the other hand, procedures S.A.1 and S.A.2 determine the start times of operations by considering, if necessary, *all* previously scheduled operations.

This explains why, for these procedures, it is possible to obtain two different semi-active schedules where the orders in which jobs are processed by the machines are the same.

The majority of the remainder of this thesis is concerned with special cases of the job-shop process in which the intermediate storage facilities are capacitated. In this chapter the general model has been described and complications which arise in its solution have been discussed.

CHAPTER 3

FLOWSHOP PROBLEMS

3.1 INTRODUCTION

A flowshop process is a special case of the general job-shop process described in the previous chapter, where now, the order in which the components of a job are executed is the same for each job. Further, it is assumed that there is a storage bin at the end of each machine into which each job goes when it is finished on that machine; in other words, all jobs follow the same path through the job-shop.

By the nature of the situation described here, it is convenient to index the machines so that each job visits them in the order 1,2,...,m. If, in addition, the bins are indexed to correspond to their associated machines, then the P and G matrices have the following special structures:

For each job j $(j=1,\ldots,n)$,

$$p_{jk} = k+1 \ ;$$

and for each operation O_{ij} $(i=1,\ldots,m;\ j=1,\ldots,n)$,

$$g_{ij} = i \ .$$

These special structures enable a drastic simplification in the general notation. Thus, the phrase,

"job j waits in bin ℓ during the interval $\{b_{kj}, a_{ij}\}$ where $g_{kj} = \ell$ and $p_{jk} = i$"

can be replaced by the much simpler,

"job j waits in bin ℓ during the interval $\{b_{\ell j}, a_{(\ell+1)j}\}$,"

where, as before, the braces indicate the alternative choices for

the nature of the intervals. This simpler notation is used throughout the chapter. Further, since each job is finished after being processed by machine m, the bin at the end of this machine is essentially the dummy location q+1 into which all completed jobs go; it therefore has unlimited capacity.

In this chapter the situations with closed waiting intervals and half-open waiting intervals are both considered. Actual scheduling algorithms are presented which can be incorporated into an implicit enumeration scheme to determine optimal schedules. Finally, for each situation, computational results are reported so that there is some indication of the size of the problems which can be solved in a reasonable amount of time.

3.2 ADDITIONAL TERMINOLOGY AND PRELIMINARY RESULTS

First the following terminology is required:

1. A *processing position matrix* PP = (pp_{ij}) is an m×n matrix for which the entry pp_{ij} specifies where operation O_{ij} is placed in the sequence of jobs processed by machine i; that is, $pp_{ij} = k$ implies that j is the k^{th} job to be processed on machine i. (3.2.1)

Clearly, since it is assumed that each job is processed by each machine exactly once, then each row of PP is a permutation of the job indices 1,2,...,n.

2. A *feasible processing position matrix* is a processing position matrix which reflects the processing orders on machines for a complete feasible schedule. (3.2.2)

Example 3.2.1

To illustrate these definitions, consider the schedule shown in figure 3.2.1. Here, the corresponding processing position

matrix is

$$PP = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} ,$$

where, for example, $pp_{22} = 1$ implies that job 2 is the *first* one processed by machine 2. Further, if the schedule shown in figure 3.2.1 is a complete feasible schedule, then PP is a *feasible* processing position matrix.

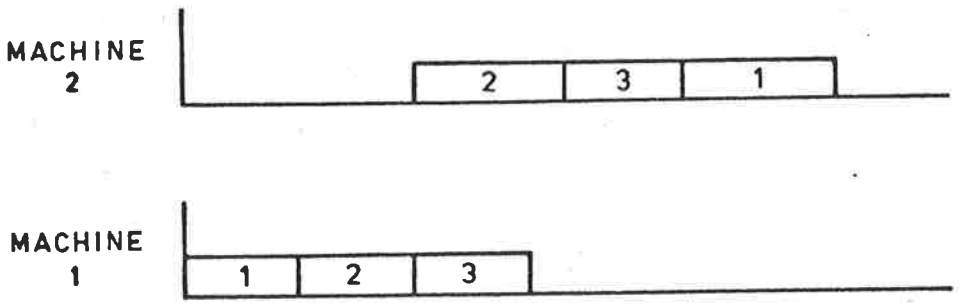


Figure 3.2.1

Although they will only be used in the context of a flowshop situation, the above definitions could clearly be applied to the general model as well.

Consider any feasible schedule for the flowshop process. For jobs which do not bypass a particular bin, the order in which they enter this bin reflects *exactly* the relative order in which they were processed by the previous machine. Similarly, the order in which they leave the bin is reflected by the order in which they are processed by the next machine. Whence, with either definition

(2.2.2) or (2.3.4) of a semi-active schedule, it is impossible to get more than one semi-active schedule corresponding to the same ordering of the jobs on the machines. Thus, corresponding to each feasible processing position matrix is a unique semi-active schedule and since there is clearly a unique processing position matrix for each semi-active schedule, this correspondence is mutual.

The first of two main results to be established in this section is:

THEOREM 3.2.1: *For the flowshop process, a processing position matrix PP is feasible if and only if for each i,j (i=1,...,m-1; j=1,...,n),*

$$pp_{ij} - pp_{(i+1)j} + \rho \leq z_i \ , \tag{3.2.3}$$

where

$$\rho = \begin{cases} 1 & \text{if waiting intervals are closed;} \\ 0 & \text{if waiting intervals are half-open.} \end{cases}$$

To underline the basic ideas needed to prove this theorem and to motivate the necessary preliminary results, consider the following outline:

OUTLINE

(only if): It is readily seen that if (3.2.3) does not hold for some i and j then PP is not feasible.

(if) : First a procedure is presented which generates a schedule from PP in such a way that, wherever possible, jobs do not *stop* in the bins; that is, if possible, both endpoints of each waiting interval are the same. This schedule satisfies all the feasibility conditions except perhaps (2.1.14) (or (2.2.1) when the waiting intervals are closed),

which requires the number of waiting intervals concurrently interacting at a bin to be no more than the capacity of the bin. Next, it is shown that if j is a job which does not stop in bin ℓ (i.e. $b_{\ell j}$ equals $a_{(\ell+1)j}$), then the number of interacting waiting intervals for bin ℓ at the instant $b_{\ell j}$ at which job j goes from machine ℓ to machine $\ell+1$ is:

$$pp_{\ell j} - pp_{(\ell+1)j} + \rho .$$

Finally, the number of waiting intervals interacting concurrently at a bin is shown to be a maximum at one of the times when a job does not stop there. Therefore, if (3.2.3) is satisfied for each i and j ($i=1,\ldots,m-1$; $j=1,\ldots,n$), the schedule produced by the procedure is feasible, since condition (2.1.14) (or (2.2.1)) is then satisfied.

A result which is required later and which may not be altogether obvious is that, for a flowshop process, it is *always* possible to obtain a schedule so that for each bin ℓ ($\ell=1,\ldots,m-1$), at least one job can go from machine ℓ to machine $\ell+1$ without stopping. For example, take the job j, say, which is processed *last* by machine ℓ. By delaying, if necessary, the start time of operation $O_{\ell j}$, it is clearly possible to make $b_{\ell j}$ equal $a_{(\ell+1)j}$, as required.

It is now necessary to present a procedure which produces a unique linear ordering σ, say, from a given processing position matrix, PP. Describing an operation as being *available* if all its predecessor operations for the same machine (as determined from

PP) and for the same job have already been included in σ, the basic idea of the procedure is to successively add to σ the available operation which corresponds to the machine with the highest index; that is, if $O_{i_1 j_1}$ and $O_{i_2 j_2}$ are available operations and $i_1 > i_2$, then $O_{i_1 j_1}$ is added to σ before $O_{i_2 j_2}$. Formally,

PROCEDURE LO

Step 0 (Initialization): Set $t_i = 1$ $(i=1,\ldots,m)$;

$\Omega = \{O_{ij} \mid i=1,\ldots,m; j=1,\ldots,n\}$;

$k = 0$;

$i = 1$.

Step 1: Find j such that $pp_{ij} = t_i$.

Step 2: If $i = 1$ or $O_{(i-1)j} \notin \Omega$, go to Step 3.

Otherwise, $i := i-1$. Go to Step 1.

Step 3: $k := k+1$, $t_i := t_i + 1$, $\sigma_k := O_{ij}$, $\Omega := \Omega - \{O_{ij}\}$.

If $i < m$, $i := i+1$. Go to Step 1.

Otherwise, if $t_m < n$ go to Step 1. Else Stop.

Example 3.2.2

Recall that the processing position matrix corresponding to the schedule in figure 3.2.1 is,

$$PP = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}.$$

Applying procedure LO to this matrix produces the linear ordering,

$$\sigma = O_{11}, O_{12}, O_{22}, O_{13}, O_{23}, O_{31}.$$

Bearing in mind the basic philosophy behind the procedure, one can more readily obtain this sequence just by inspection of the schedule in figure 3.2.1; that is, initially only O_{11} is available and so $\sigma_1 = O_{11}$. Next, only O_{12} is available and therefore $\sigma_2 = O_{12}$. Now O_{22} and O_{13} are both available and choosing the machine

with the highest index gives $\sigma_3 = O_{22}$. For each of the remaining three stages only a single operation is available and accordingly, $\sigma_4 = O_{13}$, $\sigma_5 = O_{23}$ and $\sigma_6 = O_{21}$.

A property of the linear ordering σ produced by procedure LO is:

P1 : *For job j, two successive operations O_{ij} and $O_{(i+1)j}$ are adjacent in σ if and only if there exists no job j' such that*

$$pp_{ij'} > pp_{ij} \quad \text{and} \quad pp_{(i+1)j'} < pp_{(i+1)j} .$$

Proof

Suppose at iteration k of LO, σ_k is set to O_{ij}. If $\sigma_{k+1} = O_{(i+1)j}$, then for any j' such that $pp_{(i+1)j'} < pp_{(i+1)j}$, it must follow that $pp_{ij'} < pp_{ij}$, since otherwise $O_{ij'}$ would not be in the first $k+1$ places of σ whilst its predecessor $O_{(i+1)j'}$ would. This contradicts Step 2 of the procedure.

Conversely, if there is a j' such that $pp_{ij'} > pp_{ij}$ and $pp_{(i+1)j'} < pp_{(i+1)j}$, then at iteration $k+1$, since $O_{ij'}$ has not been added to σ, then neither has $O_{(i+1)j'}$. (Step 2 again). Whence, at iteration $k+1$, $t_{i+1} \leqslant j'$ and therefore $t_{i+1} < j$; that is, $\sigma_{k+1} \neq O_{(i+1)j}$.

End of Proof.

Suppose starting times $a(\sigma_k)$ are assigned to the elements of σ using the rule:

R1 : $a(\sigma_1) = r(\sigma_1)$

$$a(\sigma_k) = \begin{cases} \max(r(\sigma_k),\ a(\sigma_{k-1})+d(\sigma_{k-1})), & \text{if } \sigma_k \text{ is an} \\ \quad \text{initial operation for some job;} \\ a(\sigma_{k-1})+d(\sigma_{k-1}), & \text{otherwise ;} \end{cases}$$

where $r(\sigma_k)$ is the earliest possible start time of an initial operation σ_k and $d(\sigma_k)$ represents the duration of operation σ_k. Then, the schedule thus obtained clearly satisfies the requirements that the components of each job are processed in the prescribed order and also that no two jobs are being processed by the same machine concurrently (i.e. feasibility conditions (2.1.11)-(2.1.13)).

Henceforth, it is convenient to describe a schedule obtained from a processing position matrix using procedure LO and rule R1 as a LOR1 schedule. Due to the fact that R1 causes at most one machine to be operating at any instant, for a LOR1 schedule, the only waiting intervals which interact are those which overlap. This holds since the critical value δ is assumed to be at most unity. Further, it should be noted that LOR1 schedules are in general *not* semi-active.

A preliminary result which follows as a direct consequence of P1 and the fact that R1 forces every operation to start as soon as the preceding operation in σ finishes or as soon as its release time allows, is:

Lemma 3.2.1: *For a LOR1 schedule, a job j stops and waits in bin i (i.e. $b_{ij} < a_{(i+1)j}$) if and only if there is a j' such that*

$$pp_{ij'} > pp_{ij} \quad \text{and} \quad pp_{(i+1)j'} < pp_{(i+1)j} \ .$$

Thus, procedure LO together with rule R1 provides a method of generating schedules with the desired features mentioned in the outline.

Prior to giving the next preliminary result, it is notationally convenient to make the following definition:

$N_\ell(t)$ = the number of waiting intervals interacting (or equivalently, overlapping) in bin ℓ at time t. (3.2.4)

Clearly, it is only necessary to look at $N_\ell(t)$ for times at which jobs finish or commence being processed by a machine adjacent to ℓ. Further, recalling that the waiting interval $\{b_{\ell j}, a_{(\ell+1)j}\}$ is to be represented by $[b_{\ell j}, a_{(\ell+1)j})$ when half-open intervals are used, $N_\ell(t)$ is a maximum at a time $b_{\ell j}$ at which some job j finishes being processed by machine ℓ. This does not necessarily assume that job j cannot bypass bin ℓ.

The next result is:

Lemma 3.2.2: *For a LOR1 schedule, if job j does not stop at bin i, then*

$$N_i(b_{ij}) = pp_{ij} - pp_{(i+1)j} + \rho \quad .$$

Proof

Firstly, $pp_{ij} - pp_{(i+1)j} \geqslant 0$, since otherwise there would be a job j' which is processed by machine i after j and by machine $i+1$ before j. If this occurred, then job j would obviously have to stop and wait in bin i.

Now, since job j does not stop in bin i, then exactly $pp_{ij} - pp_{(i+1)j}$ $(\geqslant 0)$ jobs, which are processed on machine i before job j, are processed on machine $i+1$ after job j. Thus, at time b_{ij}, there are exactly $pp_{ij} - pp_{(i+1)j}$ *other* jobs waiting in bin i. If the waiting intervals are closed, then at time b_{ij}, job j is also considered to be in bin i. Otherwise, with half-open intervals, job j actually bypasses bin i. Whence, the number of waiting intervals interacting at bin i at time b_{ij} is $pp_{ij} - pp_{(i+1)j} + \rho$.

End of Proof.

A further necessary result is:

Lemma 3.2.3: *For a LOR1 schedule, if there is a* j' *such that* $pp_{ij'} > pp_{ij}$ *and* $pp_{(i+1)j'} < pp_{(i+1)j}$, *then* $N_i(b_{ij}) \leq N_i(b_{ij'})$.

Proof

For any job $\hat{j}$ such that $pp_{i\hat{j}} < pp_{ij}$ and $pp_{(i+1)\hat{j}} < pp_{(i+1)j'}$, $O_{(i+1)\hat{j}}$ appears in σ before O_{ij}. Therefore, from R1, $\hat{j}$ is *not* waiting in bin i when O_{ij} finishes, i.e. at time b_{ij}.

Thus, any job waiting at bin i at time b_{ij} must be processed by machine $i+1$ after job j' and therefore is still waiting in bin i at time $b_{ij'}$. Whence $N_i(b_{ij}) \leq N_i(b_{ij'})$, as required. End of Proof.

A result which unifies the previous three lemmas and enables a straightforward proof of Theorem 3.2.1 to be presented is:

Lemma 3.2.4: *For a LOR1 schedule*

$$\max_{t>0} N_i(t) = \max_j (pp_{ij} - pp_{(i+1)j}) + \rho \quad (i=1,\ldots,m-1).$$

Proof

Suppose $pp_{ij^\times} - pp_{(i+1)j^\times} = \max_j (pp_{ij} - pp_{(i+1)j})$. Then,

(i) $O_{ij^\times}$ and $O_{(i+1)j^\times}$ are adjacent in σ. If this were not the case then from P1 there is a j' such that $pp_{ij'} > pp_{ij^\times}$ and $pp_{(i+1)j'} < pp_{(i+1)j^\times}$. But this would mean that $pp_{ij'} - pp_{(i+1)j'}$ is strictly *larger* than $pp_{ij^\times} - pp_{(i+1)j^\times}$, a contradiction. Thus, job $j^\times$ does not stop at bin i and therefore from lemma 3.2.2,

$$N_i(b_{ij^\times}) = pp_{ij^\times} - pp_{(i+1)j^\times} + \rho.$$

(ii) From lemma 3.2.3, $N_i(t)$ is maximized at a time b_{ij}, where for job j there is no j' such that $pp_{ij'} > pp_{ij}$ and $pp_{(i+1)j'} < pp_{(i+1)j}$. Thus, from Lemma 3.2.1, job j does not stop at bin i and so, using Lemma 3.2.2 again,

$$\max_{t>0} N_i(t) = pp_{ij} - pp_{(i+1)j} + \rho .$$

But, by definition, $pp_{ij} - pp_{(i+1)j} + \rho \leq pp_{ij^\times} - pp_{(i+1)j^\times} + \rho$ and therefore combining the above results gives,

$$\max_{t>0} N_i(t) \leq pp_{ij^\times} - pp_{(i+1)j^\times} + \rho = N_i(b_{ij^\times}) \leq \max_{t>0} N_i(t) .$$

Clearly, equality must hold throughout and the result follows.

End of Proof.

Proof of Theorem 3.2.1

(only if) : Suppose there is a job j such that for some i,

$$pp_{ij} - pp_{(i+1)j} + \rho > Z_i ,$$

and consider any schedule corresponding to PP which satisfies (2.1.11) to (2.1.13). Now, of the $pp_{ij} - 1$ jobs processed on machine i before job j, at least $pp_{ij} - pp_{(i+1)j}$ are processed on machine $i+1$ after j. Therefore, if $pp_{ij} - pp_{(i+1)j} > Z_i + \rho$, at least $Z_i + 1 - \rho$ jobs wait in bin i from before job j starts being processed on machine i until after it finishes on machine $i+1$.

If $\rho = 0$, then condition (2.1.14) is clearly violated. Also, if $\rho = 1$ then the waiting intervals are closed and job j itself is considered to occupy bin i concurrently with the other jobs and so, the feasibility condition (2.2.1) is violated. Either way, PP is *not* feasible.

(if): From Lemma 3.2.4, a LOR1 schedule obtained from PP has the property that

$$\max_{t>0} N_i(t) = \max_j (pp_{ij} - pp_{(i+1)j}) + \rho.$$

Thus, if $pp_{ij} - pp_{(i+1)j} + \rho \leq Z_i$ $(i=1,\ldots,m-1;\ j=1,\ldots,n)$, then for this LOR1 schedule

$$\max_{t>0} N_i(t) \leq Z_i \quad (i=1,\ldots,m-1) .$$

Whence, this LOR1 schedule is feasible and so, since a feasible schedule could be obtained from PP, it too is feasible.
End of Proof.

An observation associated with this result is that the existence of feasible (and therefore semi-active) schedules is independent of the job data. Given a processing position matrix and the capacities of the bins, one can decide whether there will be a corresponding feasible schedule by just checking whether or not condition (3.2.3) in Theorem 3.2.1 is satisfied for each i and j ($i=1,\ldots,m-1$; $j=1,\ldots,n$).

To enable fairly simple expressions to be used in the next main result, it is helpful to assume that $Z_i \leqslant n$ ($i=1,\ldots,m$). This causes no loss of generality since Z_i equalling n essentially provides unlimited capacity to bin i.

THEOREM 3.2.2: *For a flowshop process, the number of different feasible processing position matrices is,*

$$\prod_{i=0}^{m-1} (Z_i+1-\rho)!\,(Z_i+1-\rho)^{n-(Z_i+1-\rho)} , \qquad (3.2.5)$$

where, as before,

$$\rho = \begin{cases} 1 & \text{if waiting intervals are closed;} \\ 0 & \text{if waiting intervals are half-open.} \end{cases}$$

Proof

Given any ordering of the n jobs on machine i ($< m$), let N_i represent the number of different ways that the jobs can be processed by machine $i+1$ so that $pp_{ij} - pp_{(i+1)j} + \rho > Z_i$ for some j; that is, so that the corresponding processing position matrix is *not* feasible. Now, if

ω_k = the number of different ways that the jobs can be ordered on machine $i+1$ so that the k^{th} job to be processed by this machine is the *first* one for which $pp_{ij} - pp_{(i+1)j} + \rho > Z_i$, then,

$$N_i = \sum_{k=1}^{n} \omega_k .$$

Further, since $pp_{ij} \leqslant n$, then $pp_{ij} - k + \rho \leqslant Z_i$ whenever $k \geqslant n + \rho - Z_i$; and so

$$\omega_k = 0, \ (k = n + \rho - Z_i, \ldots, n) .$$

Whence,

$$N_i = \sum_{k=1}^{n+\rho-Z_i-1} \omega_k .$$

To obtain ω_k $(k=1,\ldots,n+\rho-Z_i-1)$, consider *all* the possibilities for the ordering of the jobs on machine $i+1$ so that a job j in any of the first $k-1$ positions satisfies $pp_{ij} - pp_{(i+1)j} + \rho \leqslant Z_i$ and so that the job in the k^{th} position satisfies $pp_{ij} - k + \rho > Z_i$. Thus, the first position of machine $i+1$'s ordering can be allocated to any of the first $Z_i + 1 - \rho$ jobs processed by machine i. (i.e. $pp_{ij} - 1 + \rho \leqslant Z_i$ iff $pp_{ij} \leqslant Z_i + 1 - \rho$). Next, the second position can be allocated to any of the first $Z_i + 2 - \rho$ jobs processed by machine i *minus* the job already allocated to the first position of machine $i+1$'s ordering. Similarly, for $h < k$, the h^{th} job to be processed by machine $i+1$ can be any of the first $Z_i + h - \rho$ jobs processed by machine i minus the $h-1$ jobs already allocated to machine $i+1$. In each case, the number of different jobs available for each of the first $k-1$ positions of machine $i+1$'s ordering is $Z_i + 1 - \rho$.

Next, the k^{th} position of the ordering on machine $i+1$ can be allocated to any of the last $n+\rho-Z_i-k$ jobs processed by machine

i (i.e. $pp_{ij} - k + \rho > Z_i$ iff $pp_{ij} > Z_i + k - \rho$); and finally, positions $k+1, k+2, \ldots, n$ of the ordering on machine $i+1$ can be filled in any order by the $n-k$ jobs not already allocated a position. Thus,

$$\omega_k = (Z_i + 1 - \rho)^{k-1} (n + \rho - Z_i - k)(n-k)! \quad ,$$

and so,

$$N_i = \sum_{k=1}^{n+\rho-Z_i-1} (Z_i + 1 - \rho)^{k-1} (n + \rho - Z_i - k)(n-k)!$$

$$= \sum_{k=1}^{n+\rho-Z_i-1} (Z_i + 1 - \rho)^{k-1} [n + 1 - k - (Z_i + 1 - \rho)](n-k)!$$

$$= \sum_{k=1}^{n+\rho-Z_i-1} (Z_i + 1 - \rho)^{k-1} (n-(k-1))! - \sum_{k=1}^{n+\rho-Z_i-1} (Z_i + 1 - \rho)^{k} (n-k)!$$

$$= n! - (Z_i + 1 - \rho)^{n+\rho-Z_i-1} (Z_i + 1 - \rho)! \; ; \qquad (3.2.6)$$

i.e. all the terms in the two summations cancel except for the first term $(k=1)$ of the first summation and the last term $(k=n+\rho-Z_i-1)$ of the second summation.

The value of N_i is independent of the *actual* ordering of the jobs on machine i. Thus for *each* ordering of the jobs on machine i there are $n!-N_i$ different ways of processing the jobs on machine $i+1$ so that $pp_{ij} - pp_{(i+1)j} + \rho \leqslant Z_i$ $(j=1,\ldots,n)$. But, only $n!-N_{i-1}$ of the possible orderings of the jobs on machine i satisfy $pp_{(i-1)j} - pp_{ij} + \rho \leqslant Z_{i-1}$ $(j=1,\ldots,n)$. Therefore, for each ordering of the jobs on machine $i-1$, there are $(n!-N_{i-1})(n!-N_i)$ different ways of processing the jobs on machine i *and* then on machine $i+1$ so that

$$\left.\begin{aligned} pp_{(i-1)j} - pp_{ij} + \rho &\leqslant Z_{i-1} \\ \text{and} \qquad pp_{ij} - pp_{(i+1)j} + \rho &\leqslant Z_i \end{aligned}\right\} \quad (j=1,\ldots,n).$$

Similarly, an inductive argument gives that, for each ordering of

the jobs on machine 1, there are $\prod_{i=1}^{m-1}(n!-N_i)$ different ways of processing the jobs on machines 2,3,...,m so that $pp_{ij} - pp_{(i+1)j} \leq Z_i$ (i=1,...,m-1; j=1,...,n). But, every ordering of the jobs on machine 1 is possible and therefore, the number of different processing position matrices is,

$$n! \prod_{i=1}^{m-1}(n!-N_i) \quad ;$$

i.e. $$n! \prod_{i=1}^{m-1}(Z_i+1-\rho)^{n-(Z_i+1-\rho)}(Z_i+1-\rho)! \quad . \qquad \text{(from (3.2.6))}$$

Now, since $Z_0 = n$, then

$$\begin{aligned}(Z_0+1-\rho)^{n-(Z_0+1-\rho)}(Z_0+1-\rho)! &= (n+1-\rho)^{\rho-1}(n+1-\rho)! \\ &= \begin{cases} n^0 n! , & \text{if } \rho = 1 ; \\ \frac{1}{n+1}(n+1)! , & \text{if } \rho = 0 . \end{cases} \\ &= n!\end{aligned}$$

Thus, the number of different processing position matrices is,

$$\prod_{i=0}^{m-1}(Z_i+1-\rho)^{n-(Z_i+1-\rho)}(Z_i+1-\rho)! \quad , \quad \text{as required.}$$

End of Proof.

The following corollary is a direct consequence of the already established one-to-one correspondence between semi-active schedules and feasible processing position matrices.

COROLLARY 3.2.1: *The number of different semi-active schedules is* $\prod_{i=0}^{m-1}(Z_i+1-\rho)^{n-(Z_i+1-\rho)}(Z_i+1-\rho)!$.

For the usual flowshop process, the number of different semi-active schedules is $(n!)^m$. The introduction of the additional feasibility constraint (2.1.14) (or (2.2.1)) has reduced this number to $\prod_{i=0}^{m-1}(Z_i+1-\rho)^{n-(Z_i+1-\rho)}(Z_i+1-\rho)!$

Example 3.2.3

As an illustration of the magnitude of this reduction, consider a situation with closed storage intervals (i.e. $\rho = 1$) where 6 jobs are to be scheduled on 3 machines with $Z_1=Z_2=2$ and $Z_0 = 6$. For this example,

$$(n!)^m \simeq 3.7 \times 10^8$$

and $$\prod_{i=0}^{m-1} (Z_i+1-\rho)^{n-(Z_i+1-\rho)} (Z_i+1-\rho)! \simeq 7.4 \times 10^5 ;$$

this latter value being about $1/500^{th}$ of the former. In general though, even the value $\prod_{i=0}^{m-1} (Z_i+1-\rho)^{n-(Z_i+1-\rho)} (Z_i+1-\rho)!$ is too large to warrant a solution process which involves direct enumeration of all semi-active schedules.

3.3 CLOSED WAITING INTERVALS

As described earlier, this situation requires every job to physically pass through *every* bin on its path through the flowshop. Job j is considered to wait in bin i during the closed interval $[b_{ij}, a_{(i+1)j}]$ and is considered as being processed by machine i during the open time interval (a_{ij}, b_{ij}).

3.3.1 The Solution Procedure

The procedure for generating semi-active schedules takes the basic structure of procedure S.A.1 presented in the previous chapter. As mentioned there, the procedure is tailored to the particular situation of interest by specifying exactly how the set of schedulable operations is determined. To avoid any confusion, a procedure tailored to the flowshop process will now be presented. As stated in the previous chapter, this procedure is a non-deterministic, dispatching algorithm. It has the property that

every maximal schedule is a *complete* semi-active schedule. When applying this procedure, the value of δ is assumed to be prescribed so that feasibility condition (2.2.1) can be checked.

Procedure S.A.1.F

Step 0 : Set $U = \{1,\ldots,n\}$;

$K = \{1,\ldots,l\}$;

$a_{1j} = \infty \quad \forall j \in U$;

$S = \phi$.

Step 1 : Determine a set Q_S of operations, where

$O_{k_j j} \in Q_S$ if (i) $j \in U$

and (ii) $\theta_S(k_j) < Z_{k_j}$.

Step 2 : $\Omega_S = \{O_{ij} \in Q_S \,|\, i = \min_{O_{h\ell} \in Q_S} h\}$. If $\Omega_S = \phi$, stop.

Step 3 : Select $O_{ij} \in \Omega_S$ and set $a_{(i+1)j} = \infty$.

Calculate integers a_{ij} and $b_{ij} = a_{ij} + d_{ij}$ such that,

(i) operation O_{ij} is scheduled after any previously scheduled operations on machine i;

(ii) none of the feasibility conditions are violated;

(iii) a_{ij} is as small as possible.

Step 4 : Add O_{ij} to S and update U and K.

Return to Step 1.

Verbally, Ω_S contains those operations in Q_S which correspond to the machine with the lowest index.

Example 3.3.1

As an illustration of how the procedure operates, consider an example with 3 jobs and 3 machines for the data indicated in Table 3.3.1. In this example, δ is chosen as unity so that all values are integer. All steps of the procedure are summarized in

Table 3.3.2 which should be self-explanatory; the complete semi-active schedule thus obtained is represented by the Gantt chart in figure 3.3.1. It should be stressed that the choice made in column four of Table 3.3.2 is completely arbitrary and if the procedure is repeated sufficiently often so that all possible combinations of choices are made, then 24 $(= \prod_{i=0}^{m-1} Z_i ! Z_i^{n-Z_i}$, since $\rho = 1)$ different semi-active schedules can be obtained. This compares very favourably with the 216 $(= (n!)^m)$ semi-active schedules which can be produced if there are no capacity restrictions on the bins.

j	r_j	d_{1j}	d_{2j}	d_{3j}
1	0	2	3	4
2	2	4	4	5
3	4	3	2	4

$Z_0=Z_3=3$; $Z_1=2$, $Z_2=1$

Table 3.3.1: Data for example 3.3.1.

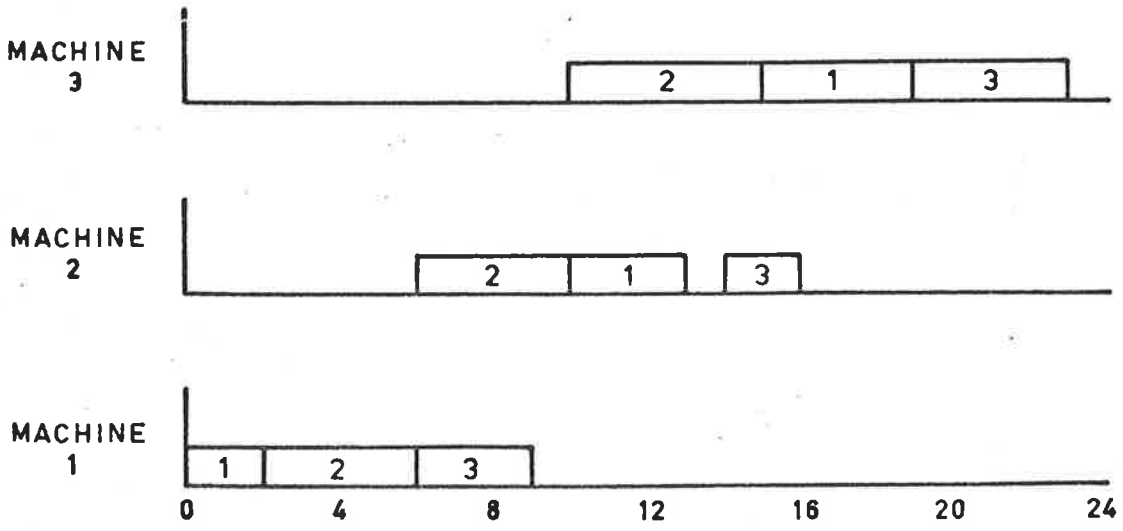


Figure 3.3.1

Possibly a better way of representing the schedule in figure 3.3.1 is to use the space-time diagram shown in figure 3.3.2. For this representation, each job is portrayed as a continuous line; the sloped segments represent the time when it is being processed by a machine and the horizontal segments

S	Q_S	Ω_S	CHOICE O_{ij}	(a_{ij},b_{ij})	COMMENT
ϕ	$\{O_{11},O_{12},O_{13}\}$	$\{O_{11},O_{12},O_{13}\}$	O_{11}	(0,2)	
$\{O_{11}\}$	$\{O_{21},O_{12},O_{13}\}$	$\{O_{12},O_{13}\}$	O_{12}	(2,6)	$O_{21}\notin\Omega_S$ as $2\neq \min_{O_{h\ell}\in Q_S} h$.
$\{O_{11},O_{12}\}$	$\{O_{21},O_{22}\}$	$\{O_{21},O_{22}\}$	O_{22}	(6,10)	$O_{13}\notin Q_S$ as $\theta_S(1)=2=Z_1$.
$\{O_{11},O_{12},O_{22}\}$	$\{O_{32},O_{13}\}$	$\{O_{13}\}$	O_{13}	(6,9)	$\theta_S(2)=1=Z_2$; $3\neq \min_{O_{h\ell}\in Q_S} h$.
$\{O_{11},O_{12},O_{22},O_{13}\}$	$\{O_{32}\}$	$\{O_{32}\}$	O_{32}	(10,15)	$\theta_S(2)=1=Z_2$.
$\{O_{11},O_{12},O_{22},O_{13},O_{32}\}$	$\{O_{21},O_{23}\}$	$\{O_{21},O_{23}\}$	O_{21}	(10,13)	
$\{O_{11},O_{12},O_{22},O_{13},O_{32},O_{21}\}$	$\{O_{31}\}$	$\{O_{31}\}$	O_{31}	(15,19)	$\theta_S(2)=1=Z_2$.
$\{O_{11},O_{12},O_{22},O_{13},O_{32},O_{21},O_{31}\}$	$\{O_{23}\}$	$\{O_{23}\}$	O_{23}	(14,16)	To satisfy (2.2.1) $b_{23}\geq a_{31}+\delta=16$.
$\{O_{11},O_{12},O_{22},O_{13},O_{32},O_{21},O_{31},O_{23}\}$	$\{O_{33}\}$	$\{O_{33}\}$	O_{33}	(19,23)	
$\{O_{11},O_{12},O_{22},O_{13},O_{32},O_{21},O_{31},O_{23},O_{33}\}$	ϕ	ϕ			STOP

Table 3.3.2: A Summary of the Computational Steps.

correspond to the time it waits in a bin. The advantage of this representation is that the number of waiting intervals interacting at a bin at any time is immediately obvious from the diagram.

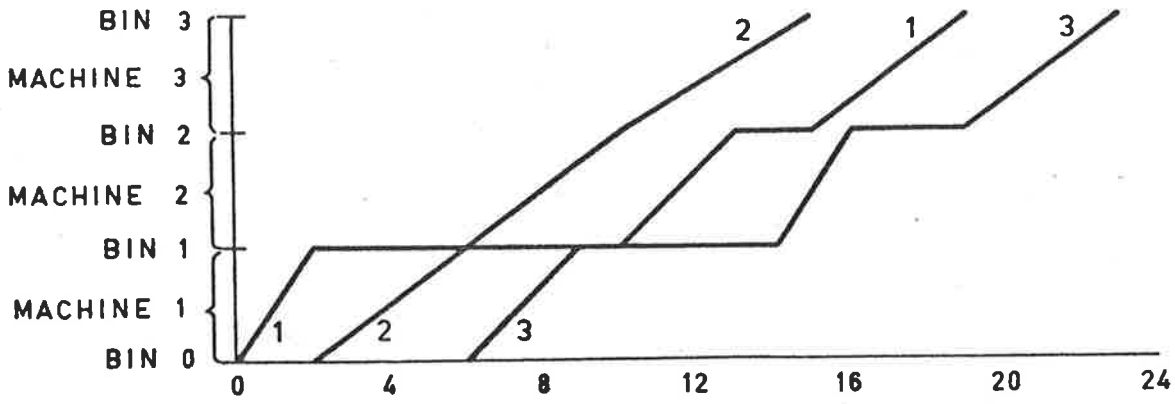


Figure 3.3.2

3.3.2 Justification for Procedure S.A.1.F

A series of lemmas are now presented which culminate in a result which essentially justifies that procedure S.A.1.F can be used to generate every complete semi-active schedule exactly once.

Lemma 3.3.1: *Whenever procedure S.A.1.F terminates, a complete semi-active schedule has been constructed.*

Proof

Since maximal schedules are semi-active in nature anyway, it is only necessary to show that, if $U \neq \phi$, then the set Q_S constructed in Step 1 is non-empty too.

So, suppose $j \in U$ and let $k_j = i$. Then either $\theta_S(i) < Z_i$, in which case $O_{ij} \in Q_S$, or else $\theta_S(i) = Z_i$. But, since $Z_\ell > 0$

for all ℓ, then $\theta_S(i) = Z_i$ means that there is a $j_1 \in U$ with $k_{j_1} = i+1$; so again, either $\theta_S(i+1) < Z_{i+1}$ and $O_{(i+1)j_1} \in Q_S$ or else $\theta_S(i+1) = Z_{i+1}$. Continue this argument until a j_r is found such that $O_{(i+1)j_r}$ is in Q_S. j_r must exist since $\theta_S(m) < n = Z_m$.

End of Proof.

Lemma 3.3.2: *If S' is the p.f.-schedule obtained from S by scheduling an operation in Ω_S, then the size of Ω_S, (i.e. $|\Omega_S|$) is independent of the choice of operation from Ω_S.*

Proof

Since all operations in Ω_S are executed on the same machine (i say), scheduling *any* one of them has the same effect on the occupancies of the storage locations; that is,

$$\begin{aligned} \theta_{S'}(i) &= \theta_S(i)+1 \\ \theta_{S'}(i-1) &= \theta_S(i-1)-1 \\ \theta_{S'}(k) &= \theta_S(k) \quad k \neq i-1,i \ . \end{aligned} \tag{3.3.1}$$

But, for $j \in U$, operation $O_{k_j j} \in Q_{S'}$ iff $\theta_{S'}(k_j) < Z_{k_j}$ (Step 1 of Procedure S.A.1.F). Therefore, since $\theta_{S'}(\ell)$ is independent of the *actual* operation scheduled at the previous stage (see (3.3.1)), so too is the number of operations in $Q_{S'}$, corresponding to each machine. Whence $|\Omega_{S'}|$ is independent of the operation chosen from Ω_S, as required.

End of Proof.

The preceding two lemmas provide some theory needed in the proof of the following result.

THEOREM 3.3.1: *The number of different complete semi-active schedules which can be produced by S.A.1.F is* $\prod_{i=0}^{m-1} Z_i^{\,n-Z_i} Z_i!$.

Proof

Let N_k^i be the *number* of choices for the operation to be scheduled at that stage of the procedure when the $(k+1)^{th}$ operation is to be scheduled on machine i. Then, from lemma 3.3.2, this number is *independent* of all previous scheduling decisions and is in fact determined by n, Z_{i-1} and k. Due to this independence of the scheduling decisions, the number of complete semi-active schedules which can be produced by S.A.1.F is

$$\prod_{i=1}^{m} \prod_{k=0}^{n-1} N_k^i \ ,$$

where from lemma 3.3.1, $N_k^i \neq 0$ $(i=1,\ldots,m;\ k=0,\ldots,n-1)$.

Moreover, each such complete schedule is *different*, since at any stage the set Ω_S contains only operations corresponding to the same machine. Thus, whenever there is a choice between two or more operations in Ω_S, the schedules which result will have these operations executed in different orders and will therefore be different schedules.

Now, suppose S is the p.f.-schedule available at some stage of the procedure and let the operations in Ω_S correspond to machine i. Then, since at most Z_{i-1} jobs can be waiting in bin i-1 to be scheduled on machine i and since procedure S.A.1.F only enables operations to be scheduled on machine i when preceding bins are fully occupied or all operations have been scheduled on earlier machines,

$$|\Omega_S| = \begin{cases} n - M_i \ , & \text{if} \quad Z_{i-1} > n - M_i \ ; \\ Z_{i-1} \ , & \text{otherwise,} \end{cases}$$

where M_i is the number of operations already scheduled on machine i. Thus,

$$N_0^i = N_1^i = \ldots = N^i_{n-Z_{i-1}} = Z_{i-1} \; ;$$

$$N^i_{n+1-Z_{i-1}} = Z_{i-1}-1 \; , \; N^i_{n+2-Z_{i-1}} = Z_{i-1}-2 \; , \ldots , N^i_{n-1} = 1 \; ;$$

and so

$$\prod_{i=1}^{m} \prod_{k=0}^{n-1} N_k^i = \prod_{i=1}^{m} (Z_{i-1})^{n-Z_{i-1}} (Z_{i-1})!$$

$$= \prod_{i=0}^{m-1} (Z_i)^{n-Z_i} Z_i ! \quad \text{as required.}$$

End of Proof.

Now from Corollary 3.2.1, for a flowshop process in which waiting intervals are closed, the number of different semi-active schedules is also $\prod_{i=0}^{m-1} (Z_i)^{n-Z_i} Z_i !$. Therefore, Theorem 3.3.1 justifies that, with appropriate "book-keeping", S.A.1.F can be used to generate *every* complete semi-active schedule *exactly* once.

In practice, the "appropriate book-keeping" takes the form of converting the procedure into a backtracking procedure as described in the previous chapter. Further, as for the general case, S.A.1.F can be made into a branch and bound procedure by allowing backtracking to occur and by discarding p.f.-schedules with lower bounds greater than the value of the current incumbent. One change to the general scheme is that it is no longer efficacious to generate a quick initial schedule using a heuristic procedure. This follows since every maximal schedule produced by S.A.1.F is a *complete* semi-active schedule and therefore, an initial schedule is obtainable *without* backtracking. If the descendant schedule retained at each stage is chosen judiciously, then the initial schedule should hopefully be close to optimal anyway.

3.3.3 Computational Results

A summary of some results obtained using the obove branch and bound procedure programmed in Fortran on a Cyber 173 is presented

in Tables 3.3.3 , 3.3.4 and 3.3.5 for the criteria Σc_j , $\Sigma w_j c_j$ and C_{max} respectively. For each combination of n and m, bin capacities were assigned values between 1 and 3 inclusive, with an emphasis on capacities of 2. All other data was determined by randomly generating values from prescribed intervals. For most of the results, the release times came from the interval [0,50], the durations came from [1,25] and weights came from [1,9]. Further, computer runs were terminated if an optimal solution was not confirmed within 4095 seconds of central processor time, a specification of T7777.

The tables have been designed to highlight some of the more important features of the results. The difference between the time required to complete a problem and the time to achieve the final schedule is due to the fact that a schedule is not known to be optimal until all further possibilities for better schedules have been exhausted. When problems were completed within 4095 seconds, the final schedule is in fact the optimal schedule. To indicate the effectiveness of the solution procedures the quantity,

$$\left(\frac{\text{Initial Value - Final Value}}{\text{Initial Value}}\right) \times 100\%$$

has been used as a measure of the relative percentage difference between the initial and final values of the criteria.

When more than one problem was solved for a particular configuration of jobs and machines, minimum, average and maximum values have been used in the tables to indicate the spread which can occur. When any of the problems was not completed in 4095 seconds, the average value of completion time has been determined using times of 4095 seconds for non-terminating problems. If, for a particular configuration, *no* problems terminated within 4095 seconds,

the single entry,

> 4095

appears in the "Time to Completion" column. Finally, the number of non-terminating problems is indicated in brackets next to the number of problems attempted.

From Tables 3.3.3, 3.3.4 and 3.3.5, it is evident that, as expected, a small increase in problem size results in a dramatic increase in computing time. This increase is more pronounced for an increase in the number of jobs rather than the number of sections. For instance, for all tables, problems with 5 jobs and 7 machines consistently require more time to solve than problems with 4 jobs and 10 machines, even though the latter problems involve the scheduling of more operations. This phenomenon can be explained by realizing that the number of possible conflicts between jobs increases linearly with the number of machines but exponentially with the number of jobs.

A feature of the tables is that for all the criteria, the problems are, in general, completed shortly after the final (usually optimal) schedule is achieved. This is not a usual occurrence with branch and bound procedures, where the optimal schedule is normally found long before the procedure terminates. One can only presume that the simplicity and specialized structure of a flowshop process has resulted in this unusual situation. A related feature of the tables is that, within a short period of time, a moderately large number of schedules is produced. Consequently, a reasonable heuristic for solving flowshop problems would be to terminate the branch and bound procedure after a prescribed time and to use the last generated schedule as the solution.

PROBLEM SIZE	NUMBER OF PROBLEMS	TIME TO COMPLETION (SECS)	TIME TO FINAL SCHEDULE (SECS)	NUMBER OF SCHEDULES	$\left(\frac{\text{INITIAL VALUE-FINAL VALUE}}{\text{INITIAL VALUE}}\right)$ 100%
4 jobs, 10 machines	10	(i) 0.247	0.037	1	0
		(ii) 2.341	2.027	5.3	5.3
		(iii) 14.614	14.320	16	20.2
5 jobs, 5 machines	20	0.063	0.019	1	0
		1.499	0.850	5.15	6.5
		9.470	4.152	13	15.1
5 jobs, 7 machines	10	0.187	0.148	3	4.5
		24.650	22.829	10.1	9.4
		179.078	168.690	20	17.0
5 jobs, 10 machines	20	0.694	0.081	2	0.7
		19.641	16.742	10.35	7.9
		171.333	157.922	24	16.3
7 jobs, 10 machines	5(1)	51.823	0.099	1	0
		1253.764	398.976	8.4	3.9
		> 4095	1720.073	13	7.0
10 jobs, 5 machines	1(1)	> 4095	2147.094	32	7.8

Table 3.3.3: Results for Σc_j criterion.

(i) Minimum value

(ii) Average value

(iii) Maximum value

PROBLEM SIZE	NUMBER OF PROBLEMS	TIME TO COMPLETION (SECS)	TIME TO FINAL SCHEDULE (SECS)	NUMBER OF SCHEDULES	$\left(\frac{\text{INITIAL VALUE-FINAL VALUE}}{\text{INITIAL VALUE}}\right)$ 100%
4 jobs, 10 machines	10	0.278	0.039	1	0
		3.789	2.880	6.6	6.7
		13.442	9.356	19	19.3
5 jobs, 5 machines	20	0.225	0.019	1	0
		1.303	0.697	7.35	9.3
		5.746	3.580	18	25.3
5 jobs, 7 machines	10	0.459	0.213	4	1.4
		9.130	3.125	10.4	9.3
		46.072	7.854	33	28.1
7 jobs, 5 machines	20	4.511	0.723	6	6.4
		204.676	111.790	24.15	14.0
		1487.161	500.360	50	26.2
7 jobs, 10 machines	$^{+}2(1)$	1777.005	0.273	3	0.5
		-	-	-	-
		> 4095	1604.108	71	21.3

Table 3.3.4: Results for $\Sigma w_j c_j$ criterion.

$^{+}$Average values are meaningless.

PROBLEM SIZE	NUMBER OF PROBLEMS	TIME TO COMPLETION (SECS)	TIME TO FINAL SCHEDULE (SECS)	NUMBER OF SCHEDULES	$\left(\frac{\text{INITIAL VALUE-FINAL VALUE}}{\text{INITIAL VALUE}}\right)$ 100%
4 jobs, 10 machines	10	0.581	0.060	2	0.5
		155.656	28.643	4.1	4.6
		1409.463	212.164	11	13.1
5 jobs, 3 machines	5	0.426	0.324	2	6.0
		5.539	3.827	5	11.7
		19.089	13.648	9	17.9
5 jobs, 5 machines	20	0.214	0.027	1	0
		53.068	28.207	4.45	8.7
		471.112	328.743	10	19.5
5 jobs, 7 machines	10	8.180	0.073	2	2.8
		445.942	195.230	5.2	8.6
		2156.496	1046.222	9	15.3
10 jobs, 4 machines	+2(2)	> 4095	0.214	3	3.1
			-	-	-
			3891.700	8	6.5

Table 3.3.5: Results for C_{max} criterion.

+Average values are meaningless.

Another feature of the tables is that the quantities,

$$\frac{\text{INITIAL VALUE - FINAL VALUE}}{\text{INITIAL VALUE}} \times 100\% \quad ,$$

are roughly comparable for all criteria. These quantities indicate that up to a 20% improvement can be obtained on the initial value of the criterion, with an average of about 8% being usual.

Comparing the completion times for the three criteria, it is clear that, for problems of the same size, the C_{max} criterion requires far more computing time than Σc_j and $\Sigma w_j c_j$, the latter two requiring similar times. In fact, it appears that problems solved with the C_{max} criterion require *fifty* times more computing time than the other two. This fact can be readily explained by realizing that, for Σc_j and $\Sigma w_j c_j$ criteria, job completion times are totalled and therefore a change in *any* of the completion times effects the value of the criterion. On the other hand, the value of the C_{max} criterion only alters when the *latest* job completion time changes. Consequently, as changes in the value of the criterion are not readily forthcoming, fewer p.f.-schedules are discarded using a bounding routine and therefore the tree of p.f.-schedules has to be searched more exhaustively. Further, this argument also explains why the average number of schedules produced for C_{max} is consistently smaller than the average number produced for the Σc_j and $\Sigma w_j c_j$ criteria

Finally, an inspection of the tables shows that problems with 7 jobs and 10 machines are about the maximum size which can be solved with the criteria Σc_j and $\Sigma w_j c_j$. For the C_{max} criterion the maximum size of a solvable problem appears to be about 5 jobs and 7 machines.

3.4 HALF OPEN WAITING INTERVALS

Adoption of half-open waiting intervals allows one to correctly model the situation where some jobs may bypass bins. Without loss of generality, job j can be considered to wait in bin i during the half-open interval $[b_{ij}, a_{(i+1)j})$ and to be processed by machine i during the time interval $[a_{ij}, b_{ij})$. The situation where job j bypasses bin i corresponds to b_{ij} equalling $a_{(i+1)j}$, in which case the interval $[b_{ij}, a_{(i+1)j})$ is empty.

3.4.1 The Solution Procedure

A procedure for generating semi-active schedules is now presented. This procedure concurs with the general procedure S.A.2 mentioned in Chapter 2 and, as stated there, is tailored to the current situation by specifying how the sets of strings of schedulable operations are determined. The analogy between the situations with closed and half-open waiting intervals prompts a procedure for generating semi-active schedules which is similar to procedure S.A.1.F presented in the previous section. In fact, the only basic difference is that strings of operations are used instead of single operations.

Procedure S.A.2.F

Step 0 : Set $U = \{1,\ldots,n\}$;

$K = \{1,\ldots,1\}$;

$a_{1j} = \infty$ for all $j \in U$;

$S = \phi$.

Step 1 : Determine a set Q_S of strings of operations, where

$\ulcorner O_{k_j j}, O_{(k_j+1)j}, \ldots, O_{(k_j+q)j} \urcorner \in Q_S$ if,

(i) $j \in U$

and (ii) $\theta_S(k_j) = Z_{k_j}, \ldots, \theta_S(k_j+q-1) = Z_{k_j+q-1}$

and $\theta_S(k_j+q) < Z_{k_j+q}$.

Step 2 : $\Omega_S = \{\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner \in Q_S \mid i = \ulcorner O_{(h-\ell)k},\ldots,O_{hk}\urcorner \in Q_S \min h\}$

If $\Omega_S = \phi$, stop.

Step 3 : Select a string $\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner$ from Ω_S and set $a_{(i+1)j}$ to infinity. Calculate integers $a_{(i-q)j}, a_{(i+1-q)j} = a_{(i-q)j}+d_{(i-q)j},\ldots,a_{ij} = a_{(i-1)j}+d_{(i-1)j}$ and $b_{ij} = a_{ij} + d_{ij}$ such that,

(i) operation $O_{(i-r)j}$ is scheduled after any previously scheduled operations on machine $i-r$ $(r=1,\ldots,q)$;

(ii) none of the feasibility conditions are violated;

(iii) $a_{(i-q)j}$ is as small as possible.

Step 4 : Add the operations $O_{(i-q)j},\ldots,O_{ij}$ to S and update U,K. Return to Step 1.

By "bending" the definition of a dispatching algorithm, as in the previous chapter, S.A.2.F can be considered as being a non-deterministic, dispatching procedure. As with procedure S.A.1.F, every maximal schedule produced by S.A.2.F is a complete semi-active schedule. Further, it can be used to generate every complete semi-active schedule exactly once, again like S.A.1.F. These facts are justified shortly, but first it is instructive to consider an example.

Example 3.4.1

Consider the problem with 4 jobs and 3 machines for the data indicated in Table 3.4.1. All the steps of the procedure are summarized in Table 3.4.2 and should require no further explanation. Finally, the complete semi-active schedule thus obtained is represented by the space-time diagram in figure 3.4.1. Two points to stress here are:

1. Unlike procedure S.A.1.F, it is not usually necessary to perform as many iterations of the algorithm S.A.2.F as there are operations. This follows since frequently more than one operation is scheduled in the same iteration.
2. In figure 3.4.1, two jobs are *not* waiting concurrently in a bin if the horizontal lines which represent waiting times *touch* but do not overlap. Observe that jobs 3,4 and 1 bypass bin 2 and job 3 bypasses bin 1.

j	r_j	d_{1j}	d_{2j}	d_{3j}
1	0	2	3	2
2	2	3	2	3
3	4	2	2	5
4	7	3	2	2

$Z_0=Z_3=4$; $Z_1=2$, $Z_2=1$.

Table 3.4.1: Data for Example 3.4.1.

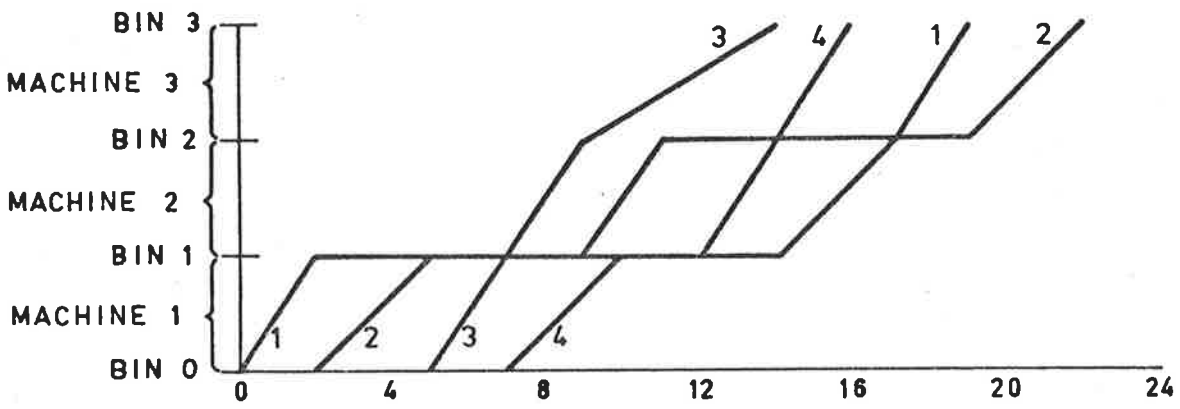


Figure 3.4.1

S	Q_S	Ω_S	CHOICE $\ulcorner o_{(i-q)j}, \ldots, o_{ij} \urcorner$	$a_{(i-q)j}, \ldots, a_{ij}, b_{ij}$	COMMENT
ϕ	$\{o_{11}, o_{12}, o_{13}, o_{14}\}$	$\{o_{11}, o_{12}, o_{13}, o_{14}\}$	o_{11}	0,2	
$\{o_{11}\}$	$\{o_{21}, o_{12}, o_{13}, o_{14}\}$	$\{o_{12}, o_{13}, o_{14}\}$	o_{12}	2,5	$o_{21} \notin \Omega_S$ as $2 \neq \ulcorner o_{(h-\ell)k}, \ldots, o_{hk} \urcorner \in Q_S{}^h \min$
$\{o_{11}, o_{12}\}$	$\{o_{21}, o_{22}, \ulcorner o_{13}, o_{23} \urcorner, \ulcorner o_{14}, o_{24} \urcorner\}$	$\{o_{21}, o_{22}, \ulcorner o_{13}, o_{23} \urcorner, \ulcorner o_{14}, o_{24} \urcorner\}$	$\ulcorner o_{13}, o_{23} \urcorner$	5,7,9	$\theta_S(1) = 2 = z_1$
$\{o_{11}, o_{12}, o_{13}, o_{23}\}$	$\{\ulcorner o_{21}, o_{31} \urcorner, \ulcorner o_{22}, o_{32} \urcorner, o_{33} \ulcorner o_{14}, o_{24}, o_{34} \urcorner\}$	$\{\ulcorner o_{21}, o_{31} \urcorner, \ulcorner o_{22}, o_{32} \urcorner, o_{33} \ulcorner o_{14}, o_{24}, o_{34} \urcorner\}$	o_{33}	9,14	$\theta_S(1) = 2 = z_1$; $\theta_S(2) = 1 = z_2$
$\{o_{11}, o_{12}, o_{13}, o_{23}, o_{33}\}$	$\{o_{21}, o_{22}, \ulcorner o_{14}, o_{24} \urcorner\}$	$\{o_{21}, o_{22}, \ulcorner o_{14}, o_{24} \urcorner\}$	o_{22}	9,11	$\theta_S(1) = z_1$
$\{o_{11}, o_{12}, o_{13}, o_{23}, o_{33}, o_{22}\}$	$\{\ulcorner o_{21}, o_{31} \urcorner, o_{32}, o_{14}\}$	$\{o_{14}\}$	o_{14}	7,10	$\theta_S(2) = z_2$; $\ulcorner o_{21}, o_{31} \urcorner, o_{32} \notin \Omega_S$ as $3 \neq \ulcorner o_{(h-\ell)k}, \ldots, o_{hk} \urcorner \in Q_S{}^h \min$ $h = 1$
$\{o_{11}, o_{12}, o_{13}, o_{23}, o_{33}, o_{22} o_{14}\}$	$\{\ulcorner o_{21}, o_{31} \urcorner, o_{32}, \ulcorner o_{24}, o_{34} \urcorner\}$	$\{\ulcorner o_{21}, o_{31} \urcorner, o_{32}, \ulcorner o_{24}, o_{34} \urcorner\}$	$\ulcorner o_{24}, o_{34} \urcorner$	12,14,16	See Step 3(i) of S.A.2.F; $\theta_S(2) = z_2$
$\{o_{11}, o_{12}, o_{13}, o_{23}, o_{33}, o_{22} o_{14}, o_{24}, o_{34}\}$	$\{\ulcorner o_{21}, o_{31} \urcorner, o_{32}\}$	$\{\ulcorner o_{21}, o_{31} \urcorner, o_{32}\}$	$\ulcorner o_{21}, o_{31} \urcorner$	14,17,19	$\theta_S(2) = z_2$
$\{o_{11}, o_{12}, o_{13}, o_{23}, o_{33}, o_{22} o_{14}, o_{24}, o_{34}, o_{21}, o_{31}\}$	$\{o_{32}\}$	$\{o_{32}\}$	o_{32}	19,22	
$\{o_{11}, o_{12}, o_{13}, o_{23}, o_{33}, o_{22} o_{14}, o_{24}, o_{34}, o_{21}, o_{31}, o_{32}\}$	ϕ	ϕ			STOP

Table 3.4.2: Summary of Computational Steps .

3.4.2 Justification for Procedure S.A.2.F

The proof that S.A.2.F can be used to generate every complete semi-active schedule exactly once requires a more complicated argument than it did for S.A.1.F, since the number of operations in the schedulable strings need not be constant. Again, a number of lemmas are used to present the argument.

Lemma 3.4.1: *Suppose at some stage of procedure S.A.2.F the current p.f.-schedule is* S. *Then, the number of strings of operations in* Q_S *is equal to the number of jobs not yet completely scheduled; that is,* $|U| = |Q_S|$.

Proof

Suppose $j \in U$ and let $k_j = i$.

If $\theta_S(i) < Z_i$, then $\ulcorner O_{ij} \urcorner \in Q_S$. On the other hand, if $\theta_S(i) = Z_i$ and $\theta_S(i+1) < Z_{i+1}$, then $\ulcorner O_{ij}, O_{(i+1)j} \urcorner \in Q_S$. Continue this type of argument until an r is found such that,

$$\theta_S(i) = Z_i, \ldots, \theta_S(i+r-1) = Z_{i+r-1}, \ \theta_S(i+r) < Z_{i+r},$$

in which case $\ulcorner O_{ij}, \ldots, O_{(i+r)j} \urcorner \in Q_S$. Such an r exists since, if $U \neq \phi$, then $\theta_S(m) < n = Z_m$. Whence, for each $j \in U$, there is a corresponding unique string of operations in Q_S; i.e. $|U| = |Q_S|$.

End of Proof.

Now, a p.f.-schedule S, available at some stage of procedure S.A.2.F, clearly satisfies the condition,

$$\theta_S(i) \leqslant Z_i \quad (i=0,\ldots,m) \ . \tag{3.4.1}$$

Any p.f.-schedule satisfying this condition will be termed a *p.p.f.-schedule* (particular partial feasible schedule). Further, for any p.p.f.-schedule S, define a *removable string of operations* to be a string $\ulcorner O_{ij}, \ldots, O_{(i+q)j} \urcorner$ in S such that

(i) $O_{(i+q)j}$ is the last *scheduled* operation for job j;

(ii) $O_{\ell j}$ is the last scheduled operation on machine ℓ $(\ell=i,\ldots,i+q)$;

(iii) $\theta_S(i-1) < Z_{i-1}$ and $\theta_S(\ell) = Z_\ell$ $(\ell=i,\ldots,i+q-1)$. (3.4.2)

Unscheduling a removable string of operations $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner$ from a p.p.f.-schedule S means that the operations $O_{ij},\ldots,O_{(i+q)j}$ are removed from S and a_{ij} is set to infinity.

Lemma 3.4.2: *Every non-empty p.p.f.-schedule S contains a removable string of operations and if any such string is unscheduled, the resulting schedule is still a p.p.f.-schedule.*

Proof

(i) Suppose O_{ij} is the last scheduled operation on machine i and for job j. Further, let $\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner$ be the string of operations for which

$$\theta_S(i-q-1) < Z_{i-q-1} ,$$

and

$$\theta_S(\ell) = Z_\ell \quad (\ell=i-q,\ldots,i-1) .$$

Clearly, such a string exists since, for non-empty S,

$$\theta_S(0) < n = Z_0 .$$

Now, if $O_{\ell j}$ is the last scheduled operation on machine ℓ $(\ell=i-q,\ldots,i)$, then, by definition (3.4.2), $\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner$ is removable and the result follows. Otherwise, since job j bypasses the fully occupied bins $i-q,\ldots,i-1$, there must exist an operation $O_{i'j'}$ which is the last scheduled operation for machine i' and for job j' and for which $i-q \leqslant i' < i$. This situation is depicted by figure 3.4.2. When this happens, repeat the argument replacing O_{ij} by $O_{i'j'}$. If this situation persists, since i' is less than i, a stage must ultimately be reached

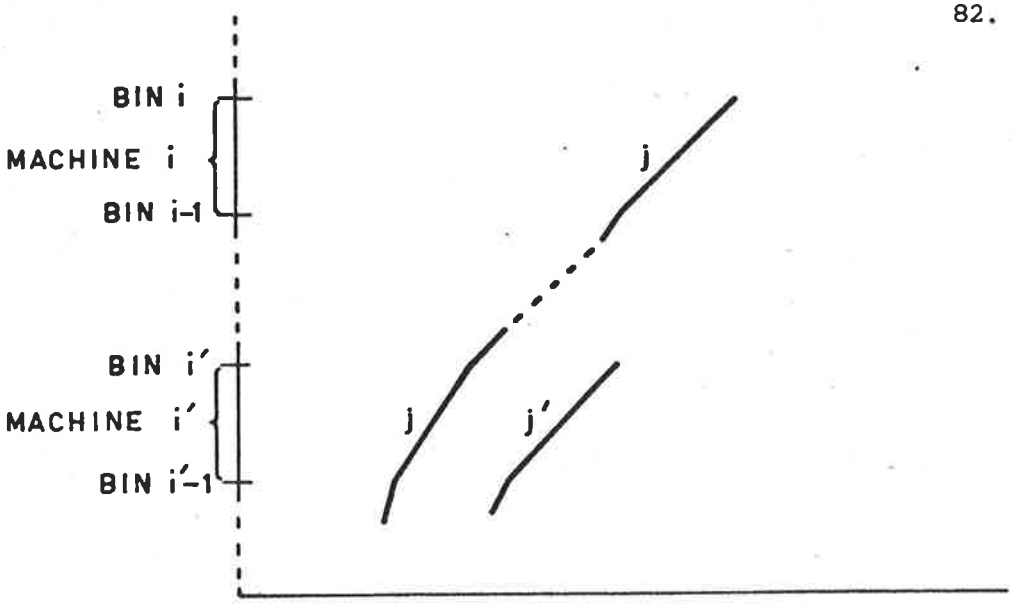


Figure 3.4.2

where $i' = 1$, in which case $\ulcorner O_{i',j'} \urcorner$ is removable, since $\theta_S(0) < n = Z_0$.

(ii) Suppose the removable string of operations $\ulcorner O_{(i-q)j}, \ldots, O_{ij} \urcorner \in S$ is unscheduled, resulting in the schedule S'.

Then,

$$\theta_{S'}(i-q-1) = \theta_S(i-q-1) + 1 ,$$

$$\theta_{S'}(i) = \theta_S(i) - 1 ,$$

and

$$\theta_{S'}(\ell) = \theta_S(\ell) \quad \text{for all} \quad \ell \neq i-q-1, i .$$

Now, since S is a p.p.f.-schedule, then

$$\theta_S(\ell) \leqslant Z_\ell , \quad (\ell=0,\ldots,m) ;$$

and, as $\ulcorner O_{(i-q)j}, \ldots, O_{ij} \urcorner$ is removable,

$$\theta_S(i-q-1) < Z_{i-q-1} .$$

Thus, $\theta_{S'}(\ell) \leqslant Z_\ell$ $(\ell=0,\ldots,m)$, as required.

End of Proof.

For a non-empty p.p.f.-schedule S, it is convenient to define,

R_S = the set of removable strings of operations. (3.4.3)

Now, every *complete* semi-active schedule is clearly a p.p.f.-schedule

and can therefore be decomposed into a sequence of strings of operations by successively unscheduling the *unique* removable string $\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner$, say, for which,

$$i = \ulcorner O_{(t-r)k},\ldots,O_{tk}\urcorner \in R_S \quad \max \; t \; . \tag{3.4.4}$$

Here, S corresponds to the p.p.f.-schedule available at any arbitrary stage of the decomposition. The uniqueness of the removable string follows from the fact that otherwise there would be two different strings $\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner$, $\ulcorner O_{(\ell-s)j'},\ldots,O_{\ell j'}\urcorner$ in R_S for which i equals ℓ. This is an impossible situation since it would require O_{ij} *and* $O_{\ell j'}$ to both be the last scheduled operation on machine i. It should be noted here that Lemma 3.4.2 justifies that the decomposition is possible, since, after any string of operations is removed, the resulting schedule S is still a p.p.f.-schedule and therefore, as R_S is non-empty, another string of operations can be removed, and so on.

Example 3.4.2

As an example of how this decomposition procedure works, consider the complete semi-active schedule produced earlier using S.A.2.F and shown in figure 3.4.1. The steps of the decomposition procedure are summarized in Table 3.4.3.

This is not a particularly good example since R_S only contains a single string of operations at each step and no choice has to be made. On the other hand, comparing the table below with Table 3.4.2 illustrates the fact that this decomposition procedure is the *exact* reverse of scheduling procedure S.A.2.F. This result will be proved shortly.

STEP	R_S	UNSCHEDULED STRING	COMMENTS
1	$\{O_{32}\}$	O_{32}	
2	$\{\ulcorner O_{21},O_{31}\urcorner\}$	$\ulcorner O_{21},O_{31}\urcorner$	
3	$\{\ulcorner O_{24},O_{34}\urcorner\}$	$\ulcorner O_{24},O_{34}\urcorner$	
4	$\{O_{14}\}$	O_{14}	$\begin{cases} O_{33}\notin R_S \text{ as } \theta_S(2) < Z_2 = 1 \\ O_{22}\notin R_S \text{ as } \theta_S(1) < Z_1 = 2. \end{cases}$
5	$\{O_{22}\}$	O_{22}	
6	$\{O_{33}\}$	O_{33}	$\begin{cases} \theta_S(2)=0 < Z_2 \Rightarrow O_{13},O_{23} \text{ are not} \\ \text{in string with } O_{33} \end{cases}$
7	$\{\ulcorner O_{13},O_{23}\urcorner\}$	$\ulcorner O_{13},O_{23}\urcorner$	
8	$\{O_{12}\}$	O_{12}	
9	$\{O_{11}\}$	O_{11}	

Table 3.4.3: The Decomposition of the Schedule in Figure 3.4.1.

Lemma 3.4.3: *For any string of operations* $\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner$ *removed from a semi-active schedule using the above decomposition procedure,*

$$pp_{(i-q-1)j} - pp_{(i-q)j} < Z_{i-q-1} \quad \text{if} \quad i-q>1\ ; \tag{3.4.5}$$

$$pp_{\ell j} - pp_{(\ell+1)j} = Z_\ell \qquad (\ell=i-q,\ldots,i-1); \tag{3.4.6}$$

$$pp_{ij} - pp_{(i+1)j} < Z_i \qquad \text{if} \quad i<m\ , \tag{3.4.7}$$

where PP is the processing position matrix corresponding to the complete schedule.

Proof

Suppose $\ulcorner O_{(i-q)j},\ldots,O_{ij}\urcorner$ is to be removed from the p.p.f.-schedule S.

(i) For S, of the first $pp_{(i-q-1)j} - 1$ jobs scheduled on machine i-q-1, at least $pp_{(i-q-1)j} - 1 - \theta_S(i-q-1)$ of them are scheduled on machine i-q. But, $O_{(i-q)j}$ is the last operation

in S to be scheduled on machine $i-q$ and therefore

$$pp_{(i-q)j} > pp_{(i-q-1)j} - 1 - \theta_S(i-q-1) \ ;$$

that is,

$$pp_{(i-q-1)j} - pp_{(i-q)j} \leqslant \theta_S(i-q-1) \ .$$

But, as $\ulcorner O_{(i-q)j},\ldots,O_{ij} \urcorner \in R_S$,

$$\theta_S(i-q-1) < Z_{i-q-1}$$

and (3.4.5) follows.

(ii) For $\ell = i-q,\ldots,i-1$, since $O_{\ell j}$ is the last operation in S scheduled on machine ℓ, then *exactly* $\theta_S(\ell)$ of the $pp_{\ell j}$ jobs scheduled on machine ℓ are *not* scheduled on machine $\ell+1$. But, $O_{(\ell+1)j}$ is the last operation in S scheduled on machine $\ell+1$ and so,

$$pp_{(\ell+1)j} = pp_{\ell j} - \theta_S(\ell) \ .$$

Again, since $\ulcorner O_{(i-q)j},\ldots,O_{ij} \urcorner \in R_S$, then

$$\theta_S(\ell) = Z_\ell \qquad (\ell = i-q,\ldots,i-1) \ ;$$

and therefore,

$$pp_{\ell j} - pp_{(\ell+1)j} = Z_\ell \ , \text{ as required in (3.4.6)} \ .$$

(iii) Finally, since O_{ij} is the last operation in S for job j and on machine i, then, of the pp_{ij} jobs scheduled on machine i, $\theta_S(i)$ are waiting in bin i. Therefore only $pp_{ij} - \theta_S(i)$ jobs are scheduled on machine $i+1$ and since $O_{(i+1)j} \notin S$, then clearly

$$pp_{(i+1)j} > pp_{ij} - \theta_S(i) \ ;$$

that is,

$$pp_{ij} - pp_{(i+1)j} < \theta_S(i) \ .$$

But, $\theta_S(i) \leqslant Z_i$, since S is a p.p.f.-schedule, and therefore,

$$pp_{ij} - pp_{(i+1)j} < Z_i \text{ , as required by (3.4.7).}$$

End of Proof.

The above proof only uses the fact that at each stage of the decomposition a removable string of operations is available. Therefore, for any decomposition procedure which successively unschedules strings of operations from a current set of removable strings, the conditions (3.4.5) to (3.4.7) hold for each string removed. Conversely, for any semi-active schedule, the same strings of operations can be obtained by just using the knowledge of the order in which the machines process the jobs; that is, for each job j (j=1,...,n) consider,

$$pp_{ij} - pp_{(i+1)j} \quad \text{for each } i \ (i=1,\ldots,m-1).$$

Then each set of adjacent operations satisfying conditions (3.4.5) to (3.4.7) form one of the required strings. For clarity, special mention should be made of strings with just one operation; that is, O_{ij} is a string by itself if *any* of the following circumstances hold:

(i) $pp_{(i-1)j} - pp_{ij} < Z_{i-1}$ and i=m ;

(ii) $pp_{ij} - pp_{(i+1)j} < Z_i$ and i=1 ; or

(iii) $pp_{(i-1)j} - pp_{ij} < Z_{i-1}$ and $pp_{ij} - pp_{(i+1)j} < Z_i$.

Example 3.4.3

To consolidate the above technique, consider again the schedule in figure 3.4.1. For this schedule,

$$PP = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \\ 3 & 4 & 1 & 2 \end{pmatrix}.$$

First consider job 1.

Here,

$$pp_{11} - pp_{21} = -3 < Z_1 ,$$

and

$$pp_{21} - pp_{31} = 1 = Z_2 ;$$

and so, the strings associated with this job are O_{11} and $\ulcorner O_{21},O_{31} \urcorner$.

For job 2,

$$pp_{12} - pp_{22} = 0 < Z_1 ,$$

and

$$pp_{22} - pp_{32} = -2 < Z_2 .$$

Therefore, *each* of the operations O_{12},O_{22} and O_{32} form strings in their own right.

Similarly, for jobs 3 and 4, it is found that the strings $\ulcorner O_{13},O_{23} \urcorner$, O_{33}, O_{14} and $\ulcorner O_{24},O_{34} \urcorner$ are obtained. As expected, the above strings of operations are exactly those obtained using the decomposition shown in Table 3.4.3.

Now, suppose an arbitrary semi-active schedule is decomposed to a sequence of K strings of operations using the procedure based on condition (3.4.4). Also, suppose the terms in the sequence are indexed from K down to 1; that is, the final string of operations unscheduled becomes the first string in the sequence. Then, the following result can be proved:

THEOREM 3.4.1: *For k=1,2,...,K there is a p.p.f.-schedule* S_k *which can be obtained by using S.A.2.F to schedule the strings indexed 1,2,...,k. (i.e. the final k strings obtained by the decomposition.)*

Proof

The proof is by induction and is accomplished by showing that, if the string of operations $\ulcorner O_{ij},\ldots,O_{(i+q)j} \urcorner$ has the index k+1 in the sequence of strings, then

$$\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner \in \Omega_{S_k} \quad (k=0,1,\ldots,K-1)\ , \tag{3.4.8}$$

where $S_0 = \phi$, the empty schedule. When (3.4.8) holds, scheduling this string produces the required p.p.f.-schedule S_{k+1}.

Now, if $\ulcorner O_{1j},\ldots,O_{rj}\urcorner$ is the final string of operations removed from the schedule, then if $r > 1$, it would mean that $Z_\ell = 0$ $(\ell=1,\ldots,r-1)$, in which case $\ulcorner O_{1j},\ldots,O_{rj}\urcorner \in \Omega_\phi$. When r equals 1, clearly $O_{1j} \in \Omega_\phi$ and so in either case (3.4.8) holds for $k=0$.

Assume that (3.4.8) holds for all k up to and including $\bar{k}$ and show that $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner \in \Omega_{S_{\bar{k}}}$, where this string has the index $\bar{k}+1$. Since $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner$ has been obtained by being unscheduled from a p.p.f.-schedule which gave $S_{\bar{k}}$, then

$$\theta_{S_{\bar{k}}}(\ell) = Z_\ell \quad (\ell=i,\ldots,i+q-1)$$

and

$$\theta_{S_{\bar{k}}}(i+q) < Z_{i+q}\ .$$

Therefore,

$$\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner \in \Omega_{S_{\bar{k}}}\ .$$

Suppose $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner \notin \Omega_{S_{\bar{k}}}$, then there must be a string of operations $\ulcorner O_{(s-r)j'},\ldots,O_{sj'}\urcorner \in \Omega_{S_{\bar{k}}}$ for which $s < i$; that is, if $s \geqslant i$ then, as $\ulcorner O_{(s-r)j'},\ldots,O_{sj'}\urcorner \in \Omega_{S_{\bar{k}}}$,

$$\theta_{S_{\bar{k}}}(\ell) = Z_\ell \quad \text{for} \quad \ell = s-r,\ldots,i+q-1\ ,$$

and so $s \geqslant i + q$. Whence, by step 2 of S.A.2.F, $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner \in \Omega_{S_{\bar{k}}}$ too, a contradiction.

Now, suppose in the original semi-active schedule, $O_{(s-r)\hat{j}}$ was the operation executed next on machine $s-r$ after those in $S_{\bar{k}}$.

Then, since

$$\theta_{S_{\bar{k}}}(\ell) = Z_\ell \quad (\ell=s-r,\ldots,s-1)$$

and

$$\theta_{S_{\bar{k}}}(s) < Z_s \ ,$$

$\ulcorner O_{(s-r)\hat{j}},\ldots,O_{s\hat{j}}\urcorner \in \Omega_{S_{\bar{k}}}$ too. If this string is scheduled next, then, for *any* completion of the resulting schedule (including the original semi-active schedule),

$$pp_{(s-r-1)\hat{j}} - pp_{(s-r)\hat{j}} < Z_{(s-r-1)} \quad \text{if} \quad s-r > 1 \ ;$$

$$pp_{\ell\hat{j}} - pp_{(\ell+1)\hat{j}} = Z_\ell \quad (\ell=s-r,\ldots,s-1);$$

and

$$pp_{s\hat{j}} - pp_{(s+1)\hat{j}} < Z_s \quad \text{if} \quad s < m \ .$$

This follows since $O_{\ell\hat{j}}$ is the next operation executed on each of the machines $s-r,\ldots,s$. But, the above conditions are just (3.4.5) to (3.4.7) and therefore $\ulcorner O_{(s-r)\hat{j}},\ldots,O_{s\hat{j}}\urcorner$ is one of the strings obtained from the decomposition. But, since $s < i$, the string $\ulcorner O_{(s-r)\hat{j}},\ldots,O_{s\hat{j}}\urcorner$ would be unscheduled *after* $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner$ (see condition (3.4.4)) and would therefore have a lower index than $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner$. This contradicts the fact that $\ulcorner O_{ij},\ldots,O_{(i+q)j}\urcorner$ has the index $\bar{k}+1$ and the result follows. End of Proof.

Thus, as strings of operations can be scheduled in exactly the reverse order to which they were unscheduled, the schedule obtained using S.A.2.F has the jobs being processed in the same order on the machines as for the original semi-active schedule. But, every schedule produced by S.A.2.F is a complete semi-active schedule (see Lemma 3.4.1) and therefore the original schedule is regained. Whence, *every* semi-active schedule can be generated

using procedure S.A.2.F. Further, if different choices are made from the set Ω_S at any stage of the algorithm, all resulting schedules are different, since the strings of operations in Ω_S all have an operation which is executed on a common machine. Thus, S.A.2.F can be used to produce every semi-active schedule exactly once.

In an analogous way to the situation with closed intervals, procedure S.A.2.F can be incorporated into a branch and bound solution scheme so that all semi-active schedules can be investigated. Again, it is not beneficial to have a heuristic procedure to determine an initial incumbent schedule since a complete semi-active schedule can be obtained without backtracking.

3.4.3 Computational Results

Using a branch and bound procedure as described above, a number of problems have been solved. The results are presented in Tables 3.4.4, 3.4.5 and 3.4.6 for the criteria Σc_j, $\Sigma w_j c_j$ and C_{max} respectively. As with the results for closed intervals, the values of release times, durations and weights were obtained by randomly sampling the intervals [0,50], [1,25] and [1,9] respectively. The bin capacities were prescribed to assume values between 0 and 2 inclusive with an emphasis on capacities of unity. Further, the computer runs were again terminated if an optimal solution was not confirmed within 4095 seconds of central processing time.

The tables presented here have the identical structure to those provided earlier for the flowshop processes with closed waiting intervals. Only some of the more elucidating quantities have been summarized in these tables. One should recall that for any problem size, the number of non-terminating problems is indicated in brackets next to the number of problems attempted. Further, for such

PROBLEM SIZE	NUMBER OF PROBLEMS	TIME TO COMPLETION (SECS)	TIME TO FINAL SCHEDULE (SECS)	NUMBER OF SCHEDULES	$\left(\frac{\text{INITIAL VALUE-FINAL VALUE}}{\text{INITIAL VALUE}}\right)$ 100%
4 jobs, 10 machines	10	(i) 0.387	0.291	3	1.5
		(ii) 6.413	5.840	29.5	33.8
		(iii) 29.535	28.989	56	47.7
5 jobs, 5 machines	20	0.103	0.021	1	0
		3.599	2.115	6.5	7.2
		9.905	9.267	20	20.3
5 jobs, 7 machines	10	1.567	1.243	17	29.9
		79.576	75.255	65.7	43.3
		335.189	312.705	98	57.2
7 jobs, 5 machines	10(1)	13.108	1.845	7	8.0
		1306.392	864.396	77.6	35.7
		> 4095	3627.710	161	50.2
7 jobs, 10 machines	1(1)	> 4095	2659.681	149	41.0

Table 3.4.4: Results for Σc_j criterion.

(i) Minimum value

(ii) Average value

(iii) Maximum value.

PROBLEM SIZE	NUMBER OF PROBLEMS	TIME TO COMPLETION (SECS)	TIME TO FINAL SCHEDULE (SECS)	NUMBER OF SCHEDULES	$\left(\frac{\text{INITIAL VALUE-FINAL VALUE}}{\text{INITIAL VALUE}}\right)$ 100%
4 jobs, 10 machines	10	1.410	0.790	4	6.4
		19.939	18.529	52.4	38.7
		72.384	70.812	91	55.1
5 jobs, 5 machines	40	0.920	0.362	9	20.3
		5.434	3.979	33.73	35.5
		22.248	20.959	58	51.2
5 jobs, 7 machines	10	0.805	0.440	8	9.9
		28.101	24.349	63.5	40.8
		83.655	68.505	99	51.4
7 jobs, 5 machines	11	22.922	20.675	14	6.9
		276.681	253.066	59.4	33.0
		1001.366	980.934	105	59.1
7 jobs, 10 machines	1(1)	> 4095	4046.254	217	43.4

Table 3.4.5: Results for $\Sigma w_j c_j$ criterion.

PROBLEM SIZE	NUMBER OF SCHEDULES	TIME TO COMPLETION (SECS)	TIME TO FINAL SCHEDULE (SECS)	NUMBER OF SCHEDULES	$\left(\frac{\text{INITIAL VALUE-FINAL VALUE}}{\text{INITIAL VALUE}}\right)$ 100%
4 jobs, 10 machines	10	6.332	0.411	2	1.4
		431.984	223.351	23.6	31.9
		3215.767	1707.314	47	44.5
5 jobs, 5 machines	40	5.235	0.021	1	0
		78.475	42.167	7.63	16.2
		342.269	295.893	23	48.8
5 jobs, 7 machines	11(1)	34.658	18.663	16	32.5
		1017.639	756.415	34.5	37.4
		> 4095	3197.537	52	43.2
7 jobs, 5 machines	$^{+}$2(2)	> 4095	138.420	11	17.4
			-	-	-
			1895.556	24	35.0
7 jobs, 10 machines	1(1)	> 4095	1668.301	36	19.1

Table 3.4.6: Results for C_{max} criterion.

$^{+}$Average values are meaningless.

problems, a completion time of 4095 seconds is used when calculating the average completion time.

Investigation of Tables 3.4.4, 3.4.5 and 3.4.6 reveals that, as for the results with closed waiting intervals, the increase in computing times is more rapid for an increase in the number of jobs rather than the number of machines. Also, it is again evident that, for all criteria, the difference between the time to complete a problem and the time taken to obtain the final schedule is, in general, relatively small. Further, the quantities

$$\left(\frac{\text{INITIAL VALUE - FINAL VALUE}}{\text{INITIAL VALUE}}\right) \times 100\%$$

are still comparable for all criteria, except now, improvements of more than 50% are not uncommon, with an average improvement of about 35% being usual.

Consistent with this last feature is the fact that the problems with half-open intervals generally require greater solution times than equivalent sized problems with closed waiting intervals. Likewise, it appears that significantly more schedules are generated when solving problems with half-open intervals. This latter feature indicates that a heuristic solution procedure, based on early termination of a branch and bound procedure, would be especially good for the case with half-open intervals.

The above differences between the results for the situations with half-open and closed waiting intervals can be explained by the fact that, for half-open intervals, jobs have the ability to bypass bins. This means that many schedules which would be infeasible if the waiting intervals were closed are now feasible, since jobs which bypass a bin do not contribute towards the number of waiting

intervals concurrently interacting there.

Comparing now Tables 3.4.4, 3.4.5 and 3.4.6, one again notices that the problems solved for the C_{max} criterion consistently require more computing time than similar problems solved with the Σc_j and $\Sigma w_j c_j$ criteria. Further, due to the larger computing times required to solve situations with half-open intervals, problems with 7 jobs and 5 machines are about the maximum size which can be solved with Σc_j and $\Sigma w_j c_j$ criteria. For the C_{max} criterion, the size of a solvable problem is still about 5 jobs and 7 machines.

CHAPTER 4

GENERATING SCHEDULES FOR A SINGLE TRACK RAILWAY LINE

4.1 INTRODUCTION

This chapter is concerned with another special case of the general problem described in Chapter 2. The model is now applied, in theory, to the situation where trains are to be scheduled on a single-track railway line. It is difficult to model an actual train scheduling situation precisely, since, for example, the movements of trains at the stations can be too complicated to consider in all their detail. Therefore, the work proposed here is only really suitable for determining approximate real-world schedules or for planning purposes. Further, the branch and bound procedure presented later in the chapter can, as expected, determine optimal schedules only for relatively small problem instances, and so, heuristic methods require some investigation if real-world problems are to be seriously considered.

The material in this chapter is presented as follows: After clarifying the notation for the current process, some preliminary results and additional terminology are presented. Next, a procedure which generates each semi-active schedule in exactly one way is given and then some computational results are reported. Finally, aspects of a real-world situation which would have to be included in the model, if it were to be used as an actual scheduling device, are discussed.

4.2 CHANGES TO THE NOTATION

The situation to be modelled is a railway line consisting of several sections separated by stations. On each section only a single train is allowed at any one time and at the stations a limited number of extra tracks are available to let trains pass each other. Further, each train is assumed to traverse the entire length of the track. The similarity of this situation to the general job-shop process should be obvious: The sections between the stations play the role of machines, the train-runs represent the jobs and the stations correspond to the storage bins. Although it is probably not strictly in keeping with the train-oriented terminology, the expression "operation" is retained to avoid confusion. Thus, *operation* O_{ij} corresponds to train j travelling on section i.

Since trains can travel in one of two directions along the railway track, this essentially corresponds to jobs being allocated to take one of two *opposing* routes through the job-shop. Thus, the situation being modelled resembles two opposing and interacting flowshop processes. Accordingly, the m sections can be indexed so that each train visits them in the order

$$1,2,\ldots,m$$

or

$$m,m-1,\ldots,1.$$

Whence, if

T_j = the section on which train j first travels, (4.2.1)

then the notation for representing the order in which the operations of a train are executed can be simplified to,

$$p_{jk} = \begin{cases} k+1 & \text{if } T_j = 1 ; \\ k-1 & \text{if } T_j = m . \end{cases} \qquad (4.2.2)$$

Further, since the stations are situated between sections, they can be indexed so that station ℓ is the one between sections ℓ and $\ell+1$. Thus, the corresponding G matrix is given by

$$g_{ij} = \begin{cases} i & \text{if } T_j = l \ ; \\ i-1 & \text{if } T_j = m \ . \end{cases} \qquad (4.2.3)$$

For this situation it is convenient to let the dummy bins 0 and q+1 (=m) represent the terminals at the end of the track. Therefore, both dummy locations are allowed to accept unstarted and finished trains.

The possibility of trains having to remain for a certain minimum time at each station, a so-called shunting time, can readily be included in the model; these are the values $Sh_{j\ell}$, briefly discussed in Chapter 2. To accommodate them into the general model, the feasibility condition (2.1.12) is altered to:

$$a_{ij} \geqslant b_{kj} + Sh_{j\ell} \quad \text{where} \quad p_{jk} = i, \ g_{k\ell} = \ell \quad \text{and} \quad T_j \neq i. \qquad (4.2.4)$$

The feasibility condition (2.1.11), which in this current context states that

$$a_{T_j j} \geqslant r_j \quad , \qquad (4.2.5)$$

is unchanged, since it is assumed that no shunting is performed at the terminals.

Since trains can *not* physically bypass stations, it is necessary to assume that all waiting intervals are closed. With this assumption, a critical value δ must be decided upon from the outset. Thus, as all data is integer, it is appropriate to henceforth let δ be unity. As seen earlier, this assumption enables the distinction between interacting and overlapping intervals to be discarded, since

interacting intervals must have at least one integer time point in common.

In summary, if $g_{kj} = \ell$ and $p_{jk} = i$, then train j waits in station ℓ during the interval $[b_{kj}, a_{ij}]$, including the endpoints b_{kj} and a_{ij}, and is considered to traverse section i during the interval (a_{ij}, b_{ij}), *not* including the endpoints. Further, including the values $Sh_{j\ell}$ into the model poses no additional problems for closed waiting intervals, since each train has to visit each station anyway. Thus, the concept of overlapping intervals and the definition of a semi-active schedule ((2.2.2)) are unchanged.

Finally, in this chapter it is convenient to represent schedules by space-time diagrams (or equivalently, "train-diagrams") rather than Gantt charts, since the interaction of trains in the stations is more easily visualized. An example of such a diagram is shown in figure 4.2.1. Here, lines with positive slope correspond to trains starting on section 1 and lines with negative slope represent trains travelling in the opposite direction. Horizontal lines correspond to the intervals when a train is waiting at a station.

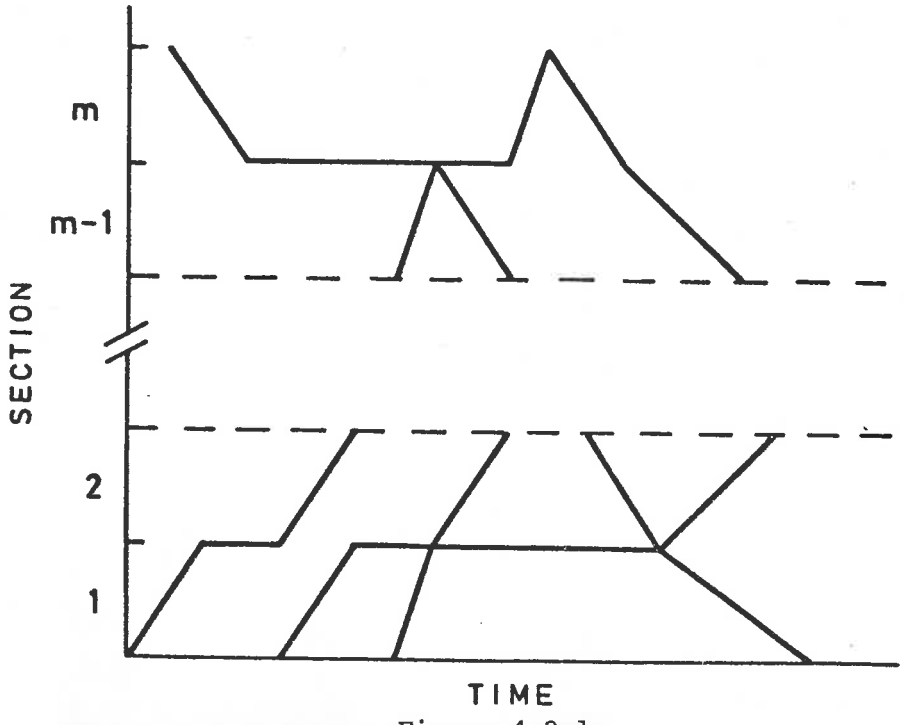


Figure 4.2.1

4.3 PRELIMINARY RESULTS

4.3.1 Terminology

Let S be a p.f.-schedule. An operation $O_{kj} \in S$ is called,

an α_S-*operation* if it is the last scheduled operation of train j (i.e. $a_{ij} = \infty$ where $p_{jk} = i$); (4.3.1)

a β_S-*operation* if it is the last scheduled operation on section k. (4.3.2)

An operation which is both an α_S and a β_S-operation is called an $\alpha_S\beta_S$-*operation*. If there is no ambiguity as to which schedule S is being referred to, the index S will be dropped and the notation is simply α-operation, β-operation etcetera.

Unscheduling an operation $O_{ij} \in S$ means removing O_{ij} from S and setting a_{ij} to infinity. This results in a new partial schedule S' which may or may not be feasible, as the feasibility condition (2.2.1) may now be violated; that is, in S' the number of trains waiting concurrently at station g_{kj}, where $p_{jk} = i$, may be more than the capacity of this station. This follows since train j occupies station g_{kj} for a longer period in S' than it did in S. An operation $O_{ij} \in S$ is called *removable*, if unscheduling it results in a partial *feasible* schedule S'. If S contains only one operation, then this operation is always considered to be removable, since the empty schedule is also regarded as a p.f.-schedule.

4.3.2 Results

Lemma 4.3.1: *For a non-empty p.f.-schedule S, an $\alpha\beta$-operation $O_{ij} \in S$ is removable if,*

$$i = T_j \ , \tag{4.3.3}$$

or if

$$\max_{O_{kh} \in S} b_{kh} \leqslant a_{ij} \quad \text{where} \quad i = p_{jk} \quad \text{and} \quad T_h = T_j \,. \tag{4.3.4}$$

Proof

Since O_{ij} is the β_S-operation for section i, then for any operation $O_{ih} \in S$ with $T_h \neq T_j$,

$$b_{ih} \leqslant a_{ij} \,.$$

Thus, if (4.3.4) holds, then *for* S, the number of trains waiting in station g_{kj} at any time after a_{ij} is one less than the number waiting *at* time a_{ij}. Whence, O_{ij} could be removed from S without increasing the maximum number of trains waiting concurrently at station g_{kj}; and, as S is feasible, the resulting schedule would be too. Also, when (4.3.3) is satisfied, operation O_{ij} is clearly removable since terminals have unlimited capacity.

End of Proof.

Every non-empty p.f.-schedule S contains an $\alpha\beta$-operation; for instance an operation with maximum b_{ij} value in S must be an $\alpha\beta$-operation. But, it is possible to prove a far stronger result, viz.:

Theorem 4.3.1: *A non-empty p.f.-schedule* S *contains a removable* $\alpha\beta$*-operation.*

Proof

Although conditions (4.3.3) and (4.3.4) in Lemma 4.3.1 are obviously not necessary for removability, it will be shown that S *always* contains an $\alpha\beta$-operation satisfying one of them.

Without loss of generality it can be assumed that a train j with $T_j = 1$ contains an $\alpha_S\beta_S$-operation and let this be O_{ij},

the i^{th} operation for train j. Suppose that neither (4.3.3) nor (4.3.4) holds, then $i > 1$ and there must be an α_S-operation $O_{(i-1)h}$ with $T_h = 1$ and $b_{(i-1)h} > a_{ij}$. $O_{(i-1)h}$ is an α_S-operation, since otherwise O_{ih} would be in S and O_{ij} could not be the β_S-operation for section i. The situation is illustrated in figure 4.3.1(a).

It will now be shown that one of the β_S-operations for the sections $1,2,\ldots,i-1$ is also an α_S-operation. There are three situations to consider:

(i) The β_S-operation on section $i-1$ corresponds to a train travelling in the same direction as train j. - This is also an α_S-operation for the same reason that $O_{(i-1)h}$ is above.

(ii) The β_S-operation on each of these sections corresponds to a train travelling in the opposite direction to train j. - The β_S-operation for section 1 is then clearly an α_S-operation.

(iii) Neither (i) nor (ii) occur; that is, there is an $i' < i-1$ such that the β_S-operation for section i', $O_{i'j'}$, say, has $T_{j'} = 1$ and the β_S-operations on sections $i'+1,\ldots,i-1$ correspond to trains travelling in the opposite direction. - Here, if $O_{(i'+1)s}$ with $T_s = m+1$ is the β_S-operation for section $i'+1$, then $O_{(i'+1)j'} \in S$ if and only if train j' travels on section $i'+1$ before train s (as in figure 4.3.1 (b)). Similarly, since $O_{i'j'}$ is the β_S-operation for section i', $O_{i's} \in S$ if and only if train s travels on section i' before train j'. (see figure 4.3.1(c)). It is clearly impossible therefore for *both* $O_{i's}$ and $O_{(i'+1)j'}$ to be in S and so at least one of $O_{i'j'}$ and $O_{(i'+1)s}$ is an α_S-operation.

Now, let

$$R = \{\alpha_s \beta_s\text{-operations on sections } 1,2,\ldots,i-1\} ;$$

and suppose $O_{i \times j \times}$ is the unique operation in R for which,

$$i^{\times} = \max_{O_{t\,h} \in R} t.$$

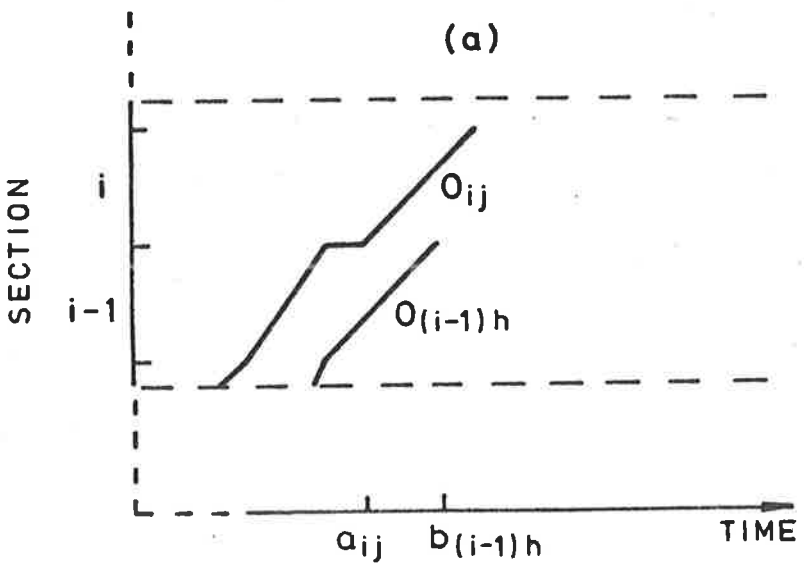
The uniqueness of $O_{i \times j \times}$ follows from the fact that there is at most one $\alpha_s \beta_s$-operation per section. Now, if $T_{j \times} = m+1$, then $O_{i \times j \times}$ satisfies (4.3.4); for, if this was not the case, then the β_s-operation for section $i^{\times}+1$, $O_{(i^{\times}+1)s}$ where $T_s = m+1$ say, would have $b_{(i^{\times}+1)s} > a_{i \times j \times}$ and would therefore be an α_s-operation (as in figure 4.3.1(d)). This would make it a member of R, contradicting the fact that

$$i^{\times} = \max_{O_{t\,h} \in R} t.$$

Alternatively, if $T_{j \times} = 1$ and $O_{i \times j \times}$ does *not* satisfy (4.3.4), then the above argument is repeated replacing $O_{i\,j}$ by $O_{i \times j \times}$. If this situation persists, then, since $i^{\times} < i$, a stage is ultimately reached where $i^{\times} = 1$, in which case (4.3.3) holds.

Thus, there is always an $\alpha\beta$-operation satisfying (4.3.3) or (4.3.4).

End of Proof.



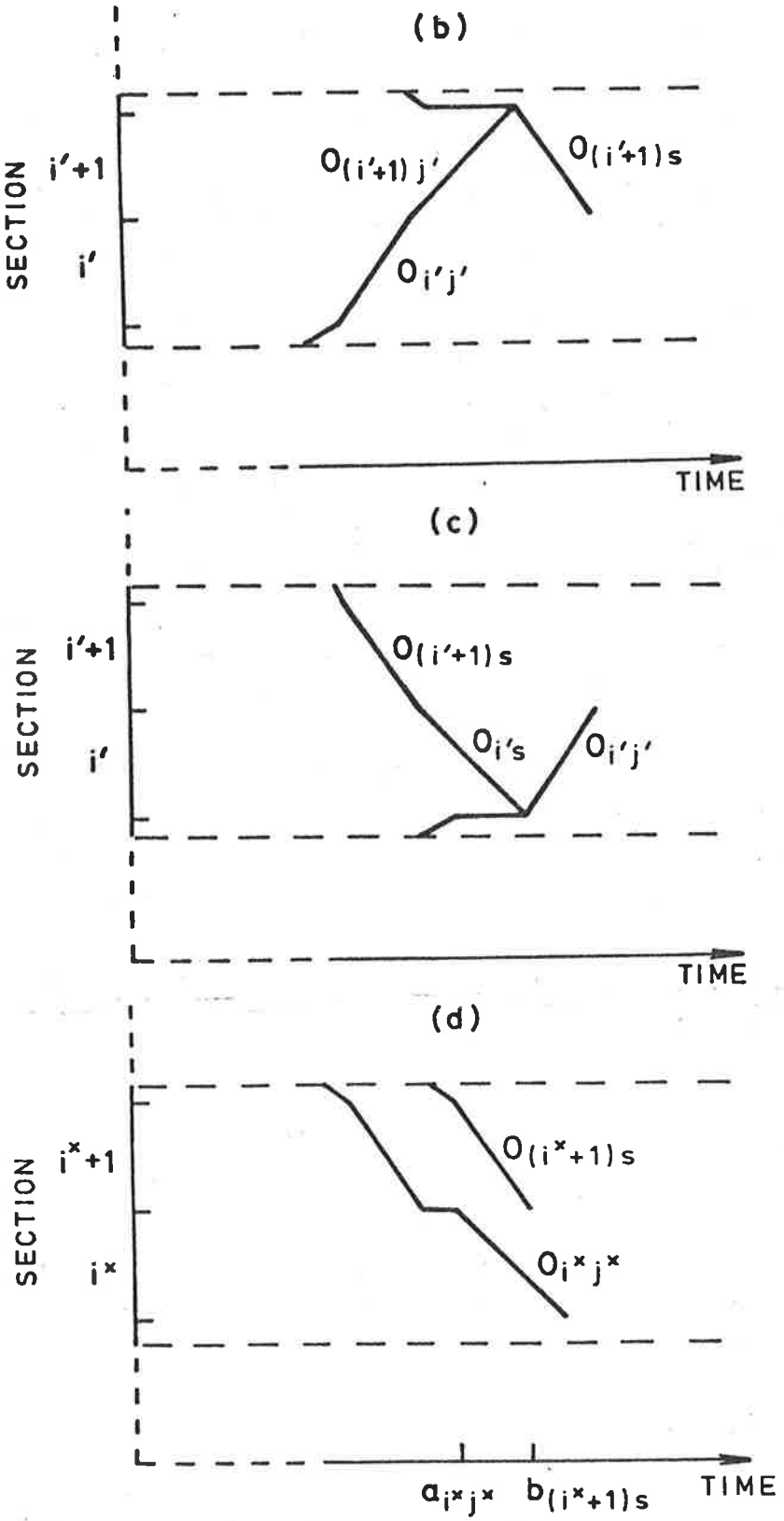


Figure 4.3.1

For convenience, a p.f.-schedule which is semi-active in nature is termed a *semi-active partial schedule*, in short a s.a.p.-schedule. Thus,

S is a *s.a.p.-schedule* if for each operation $O_{ij} \in S$, a_{ij} cannot alone be reduced without either causing one

of the feasibility conditions for operations in S to be violated or altering the order in which the scheduled operations are executed on some section. (4.3.5)

An important result concerning s.a.p.-schedules is the following:

Theorem 4.3.2: *If a removable* $\alpha\beta$*-operation in a s.a.p.-schedule* S^+ *is unscheduled, the resulting schedule* S *is again a s.a.p.-schedule.*

Proof

Suppose without loss of generality that O_{ij} with $T_j = 1$ is a removable $\alpha\beta$-operation in S^+. By unscheduling this operation, the occupancy of station i is reduced by unity for all times after and including b_{ij}. Therefore, without changing the order in which operations are executed on the sections, only an operation in S^+ which arrives at this station after b_{ij} could have its start time reduced in S. But, since O_{ij} is a β_{S^+}-operation for section i, then only an operation $O_{(i+1)h}$ with $T_h = m+1$ would be in this category. This situation is depicted in figure 4.3.2(a).

For any such operation $O_{(i+1)h}$, its start time $a_{(i+1)h}$ can be reduced only if station i was fully occupied in S^+ from b_{ij} to $b_{(i+1)h} - \delta$ (i.e. $b_{(i+1)h} - 1$), but not immediately after this time. This could only occur if there was a train which left this station at $b_{(i+1)h} - 1$. But again, since O_{ij} is the β_{S^+}-operation for section i, such a train s, say, would have to be travelling in the same direction as train j. This situation is illustrated in figure 4.3.2(b).

Now, in both schedules S and S^+, operation $O_{(i+1)h}$ is executed after operation $O_{(i+1)s}$. Whence, as $d_{(i+1)h}$ and $d_{(i+1)s}$ are positive integers,

$$a_{(i+1)s} < b_{(i+1)h} - 1.$$

This contradicts the fact that train s leaves station i *at* $b_{(i+1)h} - 1$ and therefore $a_{(i+1)h}$ can *not* be reduced.

End of Proof.

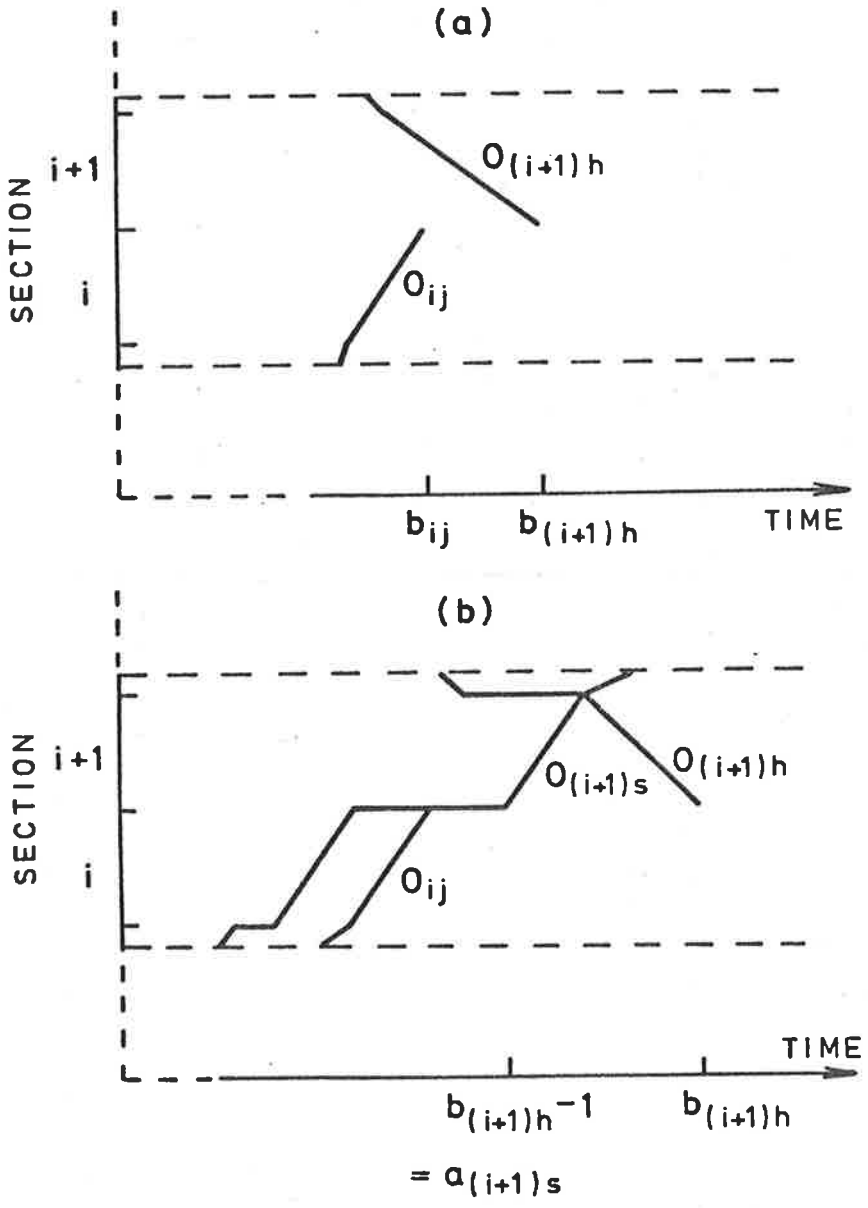


Figure 4.3.2

Clearly, the above proof is still valid for any values of δ satisfying,

$$\delta \leqslant 2 \min_{i,j} d_{ij} \quad . \tag{4.3.6}$$

But, it has already been decided in Chapter 2 that δ must be at most $\min_{i,j} d_{ij}$ to avoid certain undesirable situations and therefore (4.3.6) follows automatically; that is, the proof is valid for any *allowable* value of δ.

Further, the proof of Theorem 4.3.2 is dependent on the fact that train s traverses section i+1 before train h. Therefore, the assumption that trains traverse *all* sections is essential for the current situation. Allowing trains to enter or leave the railway line at stations other than the terminals would not be compatible with the general model in Chapter 2 and would therefore require a new approach. Accordingly, these problems are not pursued further in the thesis.

Now, if a p.f.-schedule S^+ is obtained from p.f.-schedule S by scheduling operation O_{kj}, then O_{kj} is clearly a removable $\alpha\beta$-operation in S^+. Further, if O_{rs} is any removable $\alpha_S\beta_S$-operation, then it is *still* a removable $\alpha\beta$-operation in S^+ unless,

1. it corresponds to the same train as O_{kj}; i.e., s=j; or
2. it corresponds to the same section as O_{kj}; i.e., r=k; or
3. the station g_{ts}, where $p_{st} = r$, is station g_{kj} $(=\ell)$ and $\theta_S(\ell) = Z_\ell - 1$ (i.e. $\theta_{S^+}(\ell) = Z_\ell$).

Now, if O_{rs} is an $\alpha\beta$-operation with $T_s \neq T_j$ and $g_{ts} = g_{kj}$ where $p_{st} = r$, then r equals k and condition 2. (above) holds (as illustrated in figure 4.3.3). Therefore, condition 3. can be replaced by,

3'. $T_s = T_j$, $p_{jk} = r$ and $\theta_S(\ell) = Z_\ell - 1$, where $\ell = g_{kj}$.

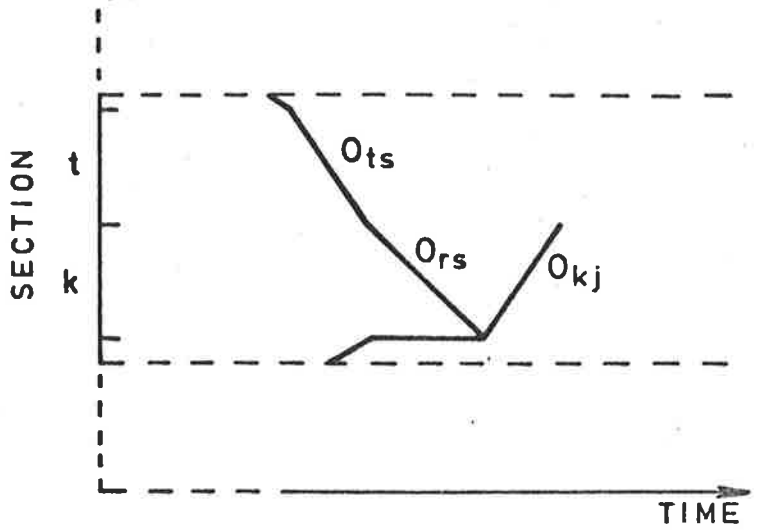


Figure 4.3.3

Thus, if

$$AB_S = \text{set of removable } \alpha\beta\text{-operations in } S, \tag{4.3.7}$$

then the following relationship holds:

$$[AB_S \cup \{O_{kj}\}] \setminus [\{\alpha_S\text{-operation of train } j,\ \beta_S\text{-operation of section } k\} \cup \{\text{if } \theta_S(\ell) = Z_\ell - 1 \text{ where } g_{kj} = \ell, \text{ the operations } O_{rs} \in S \text{ with } T_s = T_j \text{ and } r = p_{jk}\}] \subseteq AB_{S^+}, \tag{4.3.8}$$

where $A \setminus B$ are those elements in A which are not in B.

Further, the only $\alpha\beta$-operations in S which were not removable and which *might* be removable in S^+ $(= S \cup \{O_{kj}\})$ are those corresponding to trains which left from a station which was full in S, but not in S^+; clearly the only such station would be g_{tj} where $p_{jt} = k$. But, since O_{kj} is the β-operation in S^+ for section k, only an $\alpha_S\beta_S$-operation O_{th} with $T_h \neq T_j$ and $k = p_{jt}$ (as above) would be of interest. There are two situations to consider:

1. If $a_{th} \leqslant a_{kj}$,

   then unscheduling O_{th} from S^+ would cause $Z_\ell + 1$ trains to be waiting concurrently at station ℓ at time a_{kj}.

The resulting schedule would be infeasible and therefore $O_{th} \notin AB_{S^+}$. An example of this situation is shown in figure 4.3.4 where Z_ℓ equals 2.

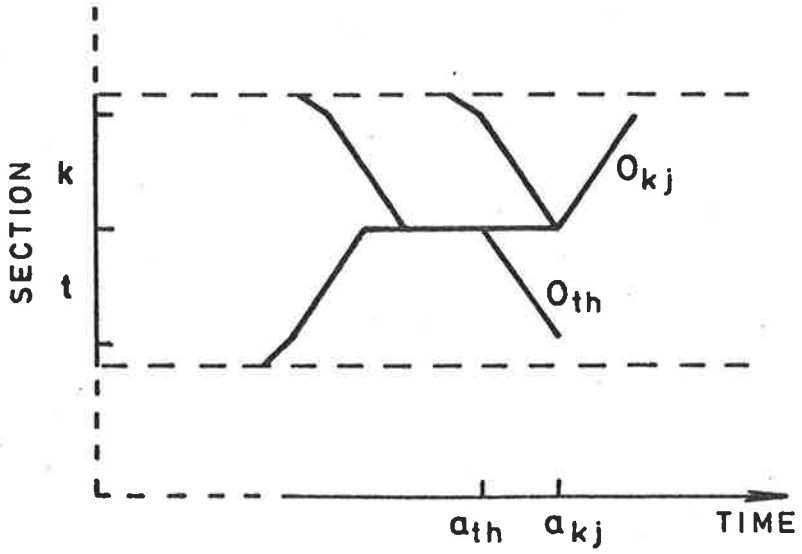


Figure 4.3.4: Schedule S^+. Unscheduling O_{th} would result in 3 (= Z_ℓ+1) trains waiting concurrently at station ℓ (= g_{tj}) at time a_{kj}.

2. Alternatively, if

$$a_{th} > a_{kj} ,$$

then for S, the number of trains waiting at station ℓ at time a_{th} is $Z_\ell + 1$, contradicting the fact that S is a p.f.-schedule. Here, the example, with Z_ℓ equalling 2, represented by figure 4.3.5 may be elucidating.

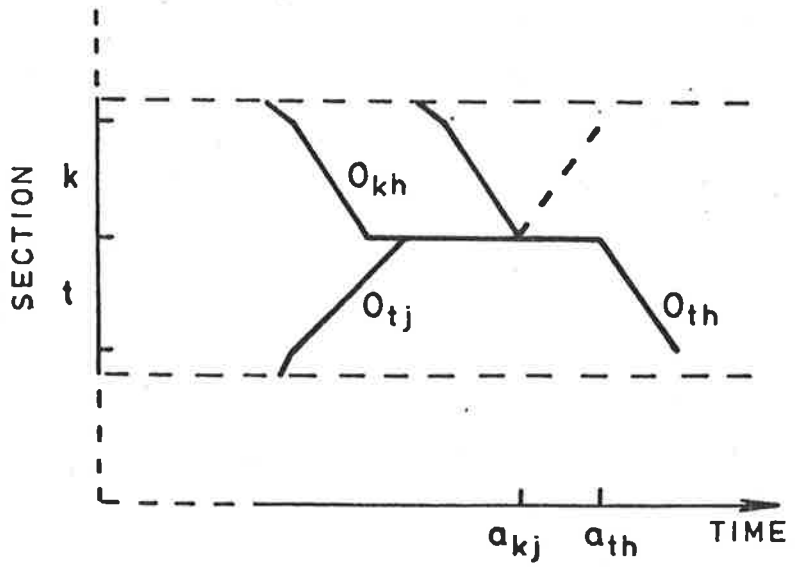


Figure 4.3.5: Schedule S. Infeasible, since the number of trains in station ℓ $(=g_{tj})$ at time a_{th} is 3 $(= Z_{\ell}+1)$.

Whence, there are *no* $\alpha\beta$-operations which were not removable in S but which are removable in S^{+}. Consequently, since no *non* $\alpha\beta$-operations in S can be $\alpha\beta$-operations in S^{+}, equality must hold in (4.3.8); that is,

$$AB_{S^{+}} = [AB_{S} \cup \{O_{kj}\}] \setminus [\{\alpha_{S}\text{-operation of train } j,\ \beta_{S}\text{-operation of section } k\} \cup \{\text{if } \theta_{S}(\ell) = Z_{\ell}-1 \text{ where } \ell = g_{kj}, \text{ the operation } O_{rs} \in S \text{ with } T_{s} = T_{j} \text{ and } r = p_{jk}\}] \quad (4.3.9)$$

This result is used in the procedure for generating s.a.p. schedules which is presented in the next section.

4.4 S.A.P.-SCHEDULE GENERATOR

Although the algorithm to be presented in this section follows the basic structure of procedure S.A.1 in Chapter 2, attention is now centred on the s.a.p.-schedule available at each stage, rather than just the maximal schedule available at termination. The reason

for this is mainly because, unlike the flowshop situation, maximal schedules need not be complete semi-active schedules. As mentioned previously, the procedure is a non-deterministic dispatching algorithm and therefore, to generate all different s.a.p.-schedules, a suitable backtracking procedure has to be added.

Procedure S.A.1.R

Step 0: Set $U = \{1,\ldots,n\}$;

$K = \{T_j \mid j \in U\}$;

$a_{T_j j} = \infty$ for all $j \in U$;

$S = \phi$.

Step 1: Determine the set Q_S of operations, where

$O_{k_j j} \in Q_S$ if (i) $j \in U$;

and (ii) $\theta_S(\ell) < Z_\ell$ where $\ell = g_{k_j j}$.

Step 2: $\Omega_S = \{O_{kj} \in Q_S \mid k \geq \max_{O_{th} \in V_S(O_{kj})} t\}$;

where $V_S(O_{kj}) = AB_S \setminus [\{\alpha_S\text{-operation of train } j\} \cup \{$if $\theta_S(\ell) = Z_\ell - 1$ where $\ell = g_{kj}$, the operations $O_{rh} \in S$ with $T_h = T_j$ and $r = p_{jk}\}]$.

If $\Omega_S = \phi$, stop.

Step 3: Select $O_{kj} \in \Omega_S$ and set $a_{ij} = \infty$ where $p_{jk} = i$.

Calculate integers a_{kj} and $b_{kj} = a_{kj} + d_{kj}$ such that

(i) operation O_{kj} is scheduled after any previously scheduled operation on section k;

(ii) none of the feasibility conditions ((4.2.4), (4.2.5), (2.1.13) and (2.2.1)) are violated;

(iii) a_{kj} is as small as possible.

Step 4: Add O_{kj} to S and update U and K.

Return to Step 1.

Clearly the set,

$$V_S(O_{kj}) \setminus \{\beta_S\text{-operation for section } k\},$$

contains those removable $\alpha\beta$-operations in S which are still removable if O_{kj} is the next operation scheduled. Therefore, if S^+ is the s.a.p.-schedule which results from S if O_{kj} is scheduled next, then from (4.3.9),

$$AB_{S^+} = [V_S(O_{kj}) \cup \{O_{kj}\}] \setminus \{\beta_S\text{-operation for section } k\}.$$

Consequently, if $k \geqslant \max_{O_{th} \in V_S(O_{kj})} t$, then

$$k = \max_{O_{th} \in AB_{S^+}} t.$$

Thus, Ω_S contains those operations in Q_S which, if scheduled next, would become the removable $\alpha\beta$-operation in S^+ corresponding to the section with the *highest index*. As will be seen shortly, this feature enables different choices from Ω_S to not only give different s.a.p.-schedules, but also to allow every semi-active schedule to be generated exactly once. But first, it is helpful to use the procedure to generate a maximal schedule.

Example 4.4.1

The presentation of this example is analogous to that used for previous algorithm-illustrating examples; that is, the data is presented in Table 4.4.1 with the steps of procedure S.A.1.R being summarized in Table 4.4.2. The resulting final schedule is shown in figure 4.4.1.

TRAIN j	T_j	r_j	d_{1j}	Sh_{1j}	d_{2j}	Sh_{2j}	d_{3j}
1	1	0	2	0	3	0	4
2	1	3	3	2	4	0	4
3	3	2	4	2	5	2	7
4	3	8	2	0	5	0	5

$Z_0=Z_3=4;$

$Z_1=2;\ Z_2=2.$

Table 4.4.1: Data for example 4.4.1.

It should be stressed that it is not always the case that a complete semi-active schedule is generated by S.A.1.R. For instance, in this example, if at the third to last stage O_{31} was chosen instead of O_{22}, then the set Ω_S at the next stage is empty and the procedure terminates without producing a complete semi-active schedule.

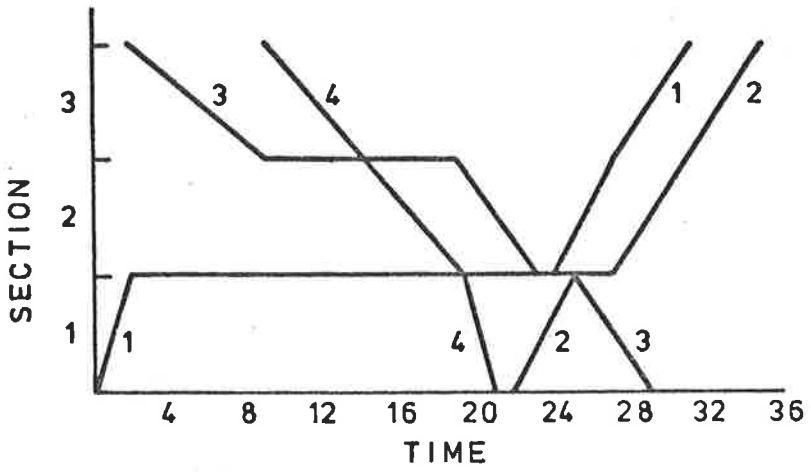


Figure 4.4.1

The important result to be justified is:

Theorem 4.4.1: *The procedure S.A.1.R generates every s.a.p.-*

S.A.P.-SCHEDULE, S	Q_S	Ω_S	CHOICE O_{kj}	(a_{kj}, b_{kj})	COMMENT
ϕ	$\{O_{11},O_{12},O_{33},O_{34}\}$	$\{O_{11},O_{12},O_{33},O_{34}\}$	O_{11}	(0,2)	
$\{O_{11}\}$	$\{O_{21},O_{12},O_{33},O_{34}\}$	$\{O_{21},O_{12},O_{33},O_{34}\}$	O_{33}	(2,9)	
$\{O_{11},O_{33}\}$	$\{O_{21},O_{12},O_{23},O_{34}\}$	$\{O_{23},O_{34}\}$	O_{34}	(9,14)	$O_{21}\notin\Omega_S$ as $O_{33}\in V_S(O_{21})$; $O_{12}\notin\Omega_S$ as $O_{33}\in V_S(O_{12})$.
$\{O_{11},O_{33},O_{34}\}$	$\{O_{12},O_{23},O_{24}\}$	$\{O_{24}\}$	O_{24}	(14,19)	$O_{21}\notin Q_S$ as $\theta_S(2)=Z_2=2$ $O_{34}\in V_S(O_{12})$ and $V_S(O_{23})$.
$\{O_{11},O_{33},O_{34},O_{24}\}$	$\{O_{21},O_{14}\}$	$\{O_{21},O_{14}\}$	O_{14}	(19,21)	$O_{12},O_{23}\notin Q_S$ as $\theta_S(1)=Z_1=2$
$\{O_{11},O_{33},O_{34},O_{24},O_{14}\}$	$\{O_{21},O_{12},O_{23}\}$	$\{O_{21},O_{12},O_{23}\}$	O_{23}	(19,24)	
$\{O_{11},O_{33},O_{34},O_{24},O_{14},O_{23}\}$	$\{O_{21},O_{13}\}$	$\{O_{21},O_{13}\}$	O_{21}	(24,27)	$O_{12}\notin Q_S$ as $\theta_S(1)=Z_1$.
$\{O_{11},O_{33},O_{34},O_{24},O_{14},O_{23},O_{21}\}$	$\{O_{31},O_{12},O_{13}\}$	$\{O_{31},O_{12},O_{13}\}$	O_{12}	(22,25)	To satisfy (2.2.1), $b_{12}-\delta \geqslant a_{21}=24$.
$\{O_{11},O_{33},O_{34},O_{24},O_{14},O_{23},O_{21},O_{12}\}$	$\{O_{31},O_{22},O_{13}\}$	$\{O_{31},O_{22},O_{13}\}$	O_{13}	(25,29)	
$\{O_{11},O_{33},O_{34},O_{24},O_{14},O_{23},O_{21},O_{12},O_{13}\}$	$\{O_{31},O_{22}\}$	$\{O_{31},O_{22}\}$	O_{22}	(27,31)	
$\{O_{11},O_{33},O_{34},O_{24},O_{14},O_{23},O_{21},O_{12},O_{13},O_{22}\}$	$\{O_{31},O_{32}\}$	$\{O_{31},O_{32}\}$	O_{31}	(27,31)	
$\{O_{11},O_{33},O_{34},O_{24},O_{14},O_{23},O_{21},O_{12},O_{13},O_{22},O_{31}\}$	$\{O_{32}\}$	$\{O_{32}\}$	O_{32}	(31,35)	
$\{O_{11},O_{33},O_{34},O_{24},O_{14},O_{23},O_{21},O_{12},O_{13},O_{22},O_{31},O_{32}\}$	ϕ	ϕ	STOP		

Table 4.4.2: Applying procedure S.A.1.R.

schedule in exactly one way.

Proof

Consider a s.a.p.-schedule S with p operations. The proof of the theorem is given by induction on p. If $p = 1$, then S can certainly be produced by S.A.1.R in exactly one way; that is, since $AB_\phi = \phi$, then $V_\phi(O_{T_j j}) = \phi$ for all j and $\Omega_\phi = Q_\phi = \{O_{T_j j} \mid j \in U\}$. So, choosing *any* operation $O_{T_j j} \in \Omega_\phi$ produces the s.a.p.-schedule $S = \{O_{T_j j}\}$, as required.

The induction step can be seen as follows. Let S^+ be *any* s.a.p.-schedule with $p+1$ operations. According to Theorem 4.3.1, $AB_{S^+} \neq \phi$; so let O_{kj} be the *unique* operation in AB_{S^+} for which

$$k = \max_{O_{th} \in AB_{S^+}} t . \qquad (4.4.1)$$

The uniqueness follows from the fact that there is at most one $\alpha\beta$-operation for any section. Further, let S be the s.a.p.-schedule which, according to Theorem 4.3.2, results if operation O_{kj} is removed from S^+. Then, S contains p operations and by the induction hypothesis can be produced by S.A.1.R in exactly one way.

Now, to show that S^+ can be generated using S.A.1.R, it is only necessary to show that O_{kj} is in Ω_S, in which case S^+ can be obtained from S in one step. The proof is as follows: Since $O_{kj} \in AB_{S^+}$, then $\theta_S(\ell) < Z_\ell$ where $\ell = g_{kj}$ and so $O_{kj} \in Q_S$. Further,

$$V_S(O_{kj}) \setminus \{\beta_S\text{-operation for section } k\} \subseteq AB_{S^+} ,$$

and so,

$$V_S(O_{kj}) \subseteq AB_{S^+} \cup \{\beta_S\text{-operation for section } k\} . \qquad (4.4.2)$$

But since, from (4.4.1),

$$\max_{O_{th} \in AB_{S^+}} t = k$$

and as,

$$\max_{O_{th} \in A} t = k,$$

where

$$A = \{\beta_S \text{ - operation for section } k\},$$

then,

$$\max_{O_{th} \in AB_{S^+} \cup A} t = k.$$

Thus, from (4.4.2),

$$\max_{O_{th} \in V_S(O_{kj})} t \leqslant k,$$

and so, $O_{kj} \in \Omega_S$, as required.

Finally, to show that S^+ cannot be generated in any other way using S.A.1.R, it is necessary to show that, if any *other* operation $O_{k'j'} \in AB_{S^+}$ is unscheduled instead of O_{kj}, then for the schedule S' so obtained, $O_{k'j'} \notin \Omega_{S'}$; that is, S^+ could not be generated from S' using S.A.1.R. So:
From the relationship ((4.3.9)) between $AB_{S'}$ and AB_{S^+}, since $O_{kj} \in AB_{S^+}$, it is in $AB_{S'}$ too. Also, as $O_{k'j'}$ and O_{kj} are both in AB_{S^+}, O_{kj} is not the $\alpha_{S'}$-operation of train j'.
Further, it cannot occur that,
$k = p_{j'k'}$, $T_j = T_{j'}$ and $\theta_{S'}(\ell') = Z_{\ell'}-1$ where $\ell' = g_{k'j'}$,
because, in that case,

$$\theta_{S^+}(\ell') = Z_{\ell'} \text{ where } \ell' = g_{k'j} \text{ and } p_{jk'} = k,$$

contradicting the fact that $O_{kj} \in AB_{S^+}$. Hence,

$$O_{kj} \in V_{S'}(O_{k'j'}) ,$$

and as O_{kj} is the unique operation for which $\max_{O_{th} \in AB_{S^+}} t$ is attained, then

$$\max_{O_{th} \in V_{S'}(O_{k'j'})} t = k > k' .$$

Whence, $O_{k'j'} \notin \Omega_{S'}$, as required.

Since S^+ is an *arbitrary* s.a.p.-schedule with p+1 operations, *every* s.a.p.-schedule can be generated in exactly one way using S.A.1.R.

End of Proof.

In particular, this result shows that every complete semi-active schedule can be obtained by scheduling a *unique* sequence of operations using procedure S.A.1.R. Moreover, given a complete semi-active schedule, (or in fact any s.a.p.-schedule), the proof of theorem 4.4.1 shows that it can be decomposed to its unique sequence of operations by successively unscheduling the removable $\alpha\beta$-operation corresponding to the section with the highest index.

Example 4.4.2

Consider again the semi-active schedule in figure 4.4.1 and work backwards as indicated above. The steps of the decomposition are summarized in Table 4.4.3. It is readily observed that the order in which the operations are unscheduled is exactly the reverse of the order in which they were scheduled using S.A.1.R.

Clearly, the sets Ω_S constructed in Step 2 of procedure S.A.1.R are independent of train-travelling, shunting and earliest

S	AB_S	$O_{kj} \in AB_S$ s.t. $k = \max_{O_{th} \in AB_S} t$	COMMENT
$\{O_{11},O_{21},O_{31},O_{12},O_{22},O_{32},O_{33},O_{23},O_{13},O_{34},O_{24},O_{14}\}$	$\{O_{32},O_{13}\}$	O_{32}	
$\{O_{11},O_{21},O_{31},O_{12},O_{22},\ O_{33},O_{23},O_{13},O_{34},O_{24},O_{14}\}$	$\{O_{31},O_{22},O_{13}\}$	O_{31}	
$\{O_{11},O_{21},\ O_{12},O_{22},\ O_{33},O_{23},O_{13},O_{34},O_{24},O_{14}\}$	$\{O_{22},O_{13}\}$	O_{22}	
$\{O_{11},O_{21},\ O_{12},\ O_{33},O_{23},O_{13},O_{34},O_{24},O_{14}\}$	$\{O_{13}\}$	O_{13}	O_{21} is an αβ-operation but not removable
$\{O_{11},O_{21},\ O_{12},\ O_{33},O_{23},\ O_{34},O_{24},O_{14}\}$	$\{O_{12}\}$	O_{12}	O_{21} is an αβ-operation but not removable
$\{O_{11},O_{21},\ O_{33},O_{23},\ O_{34},O_{24},O_{14}\}$	$\{O_{21}\}$	O_{21}	O_{14} is an αβ-operation but not removable
$\{O_{11},\ O_{33},O_{23},\ O_{34},O_{24},O_{14}\}$	$\{O_{23}\}$	O_{23}	O_{14} is an αβ-operation but not removable
$\{O_{11},\ O_{33},\ O_{34},O_{24},O_{14}\}$	$\{O_{14}\}$	O_{14}	
$\{O_{11},\ O_{33},\ O_{34},O_{24},\ \}$	$\{O_{11},O_{24}\}$	O_{24}	
$\{O_{11},\ O_{33},\ O_{34}\ \}$	$\{O_{11},O_{34}\}$	O_{34}	
$\{O_{11},\ O_{33}\ \}$	$\{O_{11},O_{33}\}$	O_{33}	
$\{O_{11}\ \}$	$\{O_{11}\}$	O_{11}	

Table 4.4.3

starting times. Thus, the number and nature of different maximal s.a.p.-schedules produced by the procedure are dependent only on the problem parameters $n_1, n_2, m; Z_1, Z_2, \ldots, Z_{m-1}$, where n_1, n_2 $(=n-n_1)$ are the number of trains travelling in each direction. Unfortunately, unlike the flowshop processes, it is not clear how to determine a formula for the number of different semi-active schedules for any set of problem parameters. The main reason for this is that it is no longer true that the number of possible orderings on "machine" i+1 is independent of the actual ordering on "machine" i.

4.5 IMPLEMENTATION AND COMPUTATIONAL RESULTS

In exactly the same way as procedure S.A.1 in Chapter 2 was changed into a branch and bound procedure, S.A.1.R can be too. In contrast to the flowshop situation, it is now advantageous to have a heuristic procedure to find an initial incumbent schedule, since, as has already been seen, many of the maximal schedules are not complete semi-active schedules. The philosophy behind the heuristic actually used in the computer programme is to successively schedule the available operation which would finish first amongst all available operations. Here, an operation is termed *available* if it can be scheduled so that the resulting partial schedule is feasible. In practice this heuristic seems to be reasonably good since, for many of the computer runs, no better schedule was produced.

A summary of the results obtained using the branch and bound algorithm programmed in Fortran on a Cyber 173 is presented in Tables 4.5.1 and 4.5.2 for the criteria Σc_j and ΣU_j respectively. These criteria are particularly relevant to a train scheduling situation because minimizing Σc_j is equivalent to minimizing the total delay and minimizing ΣU_j means that the number of tardy

trains is kept to a minimum.

For the results in Tables 4.5.1 and 4.5.2, computer runs were generally terminated if an optimal solution was not obtained within 4095 seconds of central processor time. Further, although it is not essential, an attempt was made to use data which reflected certain features of a reasonably realistic situation. For example, the ratio of train speeds between any two trains was kept roughly constant for each section and the capacities of the stations were kept small in relation to the number of trains. As for previous situations, this latter requirement is almost mandatory for our purposes, since the procedure has been designed to exploit the capacity restrictions at the stations and clearly, if this restriction is not severe, then only a small number of schedules are infeasible.

The tables have been designed to provide the more important aspects of the results. The times taken to obtain initial incumbent schedules (i.e. time to first schedule) have been included to indicate the speed at which the heuristic works. These times are in fact *very* small and are roughly constant for any problem size; that is, an initial incumbent solution is normally obtained very quickly. Further, for any problem instance, the time to achieve the optimal solution is clearly an important quantity, as is the time taken to complete the problem. As mentioned previously, these times differ, since a schedule is not known to be optimal until all further possibilities for better schedules have been exhausted. Further, knowledge of the number of schedules produced by the procedure gives an indication of how good the initial incumbent solution has been and also how effective the bounding routine is.

For problem sizes for which not all problems could be solved within 4095 seconds, the number which were not completed within this time is indicated in brackets next to the number of problems

PROBLEM SIZE	NUMBER OF PROBLEMS	TIME TO COMPLETION (SECS)	TIME TO FIRST SCHEDULE (SECS)	TIME TO OPTIMAL (SECS)	NUMBER OF SCHEDULES
5 trains, 3 sections	10	(i) 0.285	0.015	0.015	1
		(ii) 15.417	0.0156	3.275	3.7
		(iii) 83.919	0.017	14.641	8
5 trains, 6 sections	5	13.948	0.023	0.023	1
		126.996	0.024	42.141	5
		362.742	0.025	103.852	11
5 trains, 8 sections	5	6.688	0.029	0.030	1
		545.960	0.030	62.328	3.2
		1753.922	0.031	279.478	10
7 trains, 3 sections	9	23.810	0.024	0.024	1
		429.586	0.0249	169.346	7.78
		1904.376	0.029	807.186	21
10 trains, 3 sections	11(10)	27.362	0.035	0.038	1
		-	0.0376	-	1
		>4095	0.041	-	1

Table 4.5.1: Results for Σc_j criterion.

(i) Minimum value

(ii) Average value

(iii) Maximum value.

PROBLEM SIZE	NUMBER OF PROBLEMS	TIME TO COMPLETION (SECS)	TIME TO FIRST SCHEDULE (SECS)	TIME TO OPTIMAL (SECS)	NUMBER OF SCHEDULES
5 trains, 3 sections	10	0.178 2.343 7.383	0.020 0.0213 0.024	0.020 0.780 3.424	1 1.9 3
5 trains, 6 sections	5	1.093 37.890 79.116	0.023 0.0232 0.024	0.023 15.792 47.498	1 2.4 3
5 trains, 8 sections	5	0.031 133.551 316.832	0.028 0.031 0.035	0.028 94.466 295.944	1 2.4 4
7 trains, 3 sections	9	8.111 54.301 150.751	0.020 0.0232 0.025	0.514 31.828 79.517	2 3.11 4
10 trains, 3 sections	11(8)	290.814 - >4095	0.036 0.0373 0.038	290.607 - -	1 2.55 6

Table 4.5.2: Results for ΣU_j criterion.

attempted. Finally, minimum, average and maximum values have been used to indicate the spreads that can occur.

From Tables 4.5.1 and 4.5.2, it is again evident that a small increase in problem size results in a large increase in computing time. Analogously to the results for the flowshop problems, the increase is more pronounced for an increase in the number of trains rather than the number of sections. For example, for the second and last configurations for both tables, the total number of operations to be scheduled is thirty; but for 5 trains and 6 sections, no problem took more than 363 seconds to solve, whereas for 10 trains and 3 sections, most problems could not be completed within 4095 seconds.

Comparing the two criteria, it appears that for problems of similar size, it is quicker to solve one for the ΣU_j criterion rather than for Σc_j, the implication being that fewer p.f.-schedules are investigated for the former criterion. This result clearly follows from the fact that ΣU_j can only assume a small number of values (i.e. at most n+1), whereas Σc_j can take many more. Thus, a decrease of one in the value of ΣU_j is a significant drop whereas this is generally not the case with Σc_j. Therefore, one would expect that the heuristic for generating an initial incumbent solution and also the routine for finding lower bounds to be more effective for ΣU_j than for Σc_j. Also, as borne out by the results, the inflexibility of the ΣU_j criterion has caused fewer complete schedules to be generated than for Σc_j. The one apparent exception to this is the 10 train, 3 section configuration, where more schedules were generated for the ΣU_j criterion. This is readily explained by the fact that within the 4095 second time limit, the tree of p.f.-schedules would be explored to a greater depth for

ΣU_j than for Σc_j .

Finally, one can see that the problems which can be solved using the proposed branch and bound procedure are relatively small in size. Not even problems involving 10 trains and 3 sections appear to be amenable to this solution method. Consequently, heuristic techniques would certainly need to be investigated if larger problems are to be considered.

4.6 PRACTICAL APPLICATIONS

It is clear that a more detailed model of a single track railway line would be required, if it were to be used by decision makers in a real-world situation. Even if such a model was available, its use would be of dubious worth as an on-line scheduling tool since, as has been seen previously, large amounts of computer time would usually have to be expended to obtain an optimal solution. Two features of the real-life situation which may in fact make problems tractable by an implicit enumeration scheme are:

1. The data may possess structures which enable strong bounds to be calculated for p.f.-schedules; and
2. Additional operating restrictions may reduce the number of schedules which have to be investigated.

The savings in solution time accruing to these features would still probably not enable an on-line scheduling process to be contemplated; but they could conceivably provide a viable scheduling tool for planning purposes. Further, it is more appropriate to use these models in a planning capacity, since it would be unlikely that a sufficiently detailed model of reality could be achieved anyway. Moreover, a model which could be applied to a number of railway systems would generally be more beneficial than a model which is so

specific that it pertains to only one line.

4.6.1 Structured Data

Examples of the structures which real-world data may possess are:

1. Trains which are faster on one section than other trains are *as fast* as the other trains on all sections;
2. Headways between trains going in the same direction are large - especially between trains of the same type;
3. Longer shunting duties are performed by slower trains; that is, a slow goods train would normally be required to do more shunting than a fast freight train, for example;
4. Deadlines for slow trains are less restrictive than for faster trains.

Tight lower bounds can be computed by completing p.f.-schedules in such a way that the above structures are taken into account. For example, if a conflict occurs between two trains travelling in the same direction, scheduling the slower train first would generally result in a schedule which was worse than one obtained by scheduling the faster one first. Also, the large headways between trains travelling in the same direction implies that most of the conflicts occur between trains travelling in opposite directions. Therefore, when deciding which train to schedule next on a section, the number of realistic choices is comparatively small and all other choices would generally result in poor p.f.-schedules.

In the general model, none of these structures for the data has been assumed. Their introduction can clearly enable the choices of operation at each stage of the branch and bound solution procedure to be categorized as potentially good or bad. Thus, by only considering different sequences of the potentially good choices,

lower bounding routines could be developed which take into account the capacity restrictions at the stations. The bounds thus obtained would conceivably be fairly good.

One drawback of having large headways between trains travelling in the same direction is that the capacity restrictions at the stations rarely effect the feasibility of a schedule; that is, only occasionally would a situation occur where the number of trains waiting at a station was greater than the capacity of the station. Therefore, in some respects, it may be inappropriate to attempt to apply the general model to actual train scheduling situations.

4.6.2 Additional Restrictions

Features of a real-world situation which are not included in the model for a single-track railway line are:

1. Some trains are too long to be accommodated at some stations. In some circumstances, a long slow-goods train may be split and placed in two sidings, especially if the alternative entails the delaying of a passenger train;
2. Not all trains need to traverse the entire length of the track; some may enter or leave via branch lines;
3. Not all sidings at a station are identical. In particular, a distinction should at least be made between the sidings and the main track;
4. A switching time may be required between trains leaving a station and other trains arriving there. This time is the time required to physically switch the points plus some prescribed safety margin.

Item 1 above would cause more p.f.-schedules to be discarded than with the current model. Item 2 can not be included in our general model since many of the theoretical results are dependent

on the fact that the trains traverse all sections. This situation corresponds to jobs not visiting all the machines. Items 3 and 4 would correspond to additional feasibility conditions which would have to be satisfied by p.f.-schedules.

The following points describe ways in which some real-world situations can be adequately incorporated into the general train-scheduling model of the earlier sections.

1. All necessary stops, such as shunting duties, crew changes, locomotive changes and passenger stops can be included in the one quantity $Sh_{j\ell}$.
2. Signals are frequently placed in the middle of a long section, allowing it to be treated as two sections. For our model this would correspond to having a station with a capacity of unity.

4.6.3 Other Considerations

Criteria

Discussions with railway management have indicated that an important optimizing criterion is to minimize the number of late trains, where here, a train may be considered as being *late* if an extra crew is required to complete the journey. Symbolically, this corresponds to the ΣU_j criterion.

Another important criterion is to minimize the aggregate weighted delay (equivalent to $\Sigma w_j c_j$), where the weights indicate the priorities of the trains. For example, passenger trains should have large weights associated with them compared to slow goods trains; that is, passengers will not tolerate even small delays, whilst slow goods trains can be run with little regard to delays.

In reality, the above idea can be varied slightly by allowing weights to be dependent on accumulated delays. In this way, the longer a train is delayed the higher its priority (i.e. weight)

becomes. Thus, as a train gets closer to its dead-line, it can be arranged for its priority to increase dramatically so that there is a barrier to it being late.

Uses as a planning tool

As a planning tool, some areas where a more specialized model could possibly be employed are:

1. Investigating the consequences of running longer, less frequent trains;
2. Determining the effect that extending sidings or increasing the number of sidings at a station would have on the current time-tables;
3. Evaluating the capacity of the track;
4. Investigating the effect of increasing or decreasing train running times.

Railway management have indicated that the ability to satisfactorily investigate these alternatives would be a useful contribution to the improvement of any system.

An additional feature that they indicated would be useful is for a model to calculate most of the data itself. Thus, from a knowledge of the horsepowers of locomotives, the lengths and weights of the trains, the gradients of the track, etcetera, the durations d_{ij} could be calculated for any problem instance. Further, it is a relatively straight-forward task to arrange for the output from the model to be visual in the form of train diagrams.

Heuristics

Even with the additional features described above, it is clear that, within a reasonable amount of time, optimal solutions could be achieved only for systems of moderate size. Thus, heuristics

would need to be investigated so that near optimal (hopefully) schedules can be obtained quickly.

One obvious heuristic is to terminate the suggested branch and bound procedure after a specified time and to use the current incumbent schedule as the solution. This tactic is based on the fact that, in the procedure, a complete schedule is generated only if it is better than any previously generated schedule. This approach would probably be reasonably fruitful since, as for most implicit enumeration algorithms, the optimal schedule is usually generated long before it is known to be optimal.

Simulation

Another way of obtaining quick solutions is by the use of simulation. To date, a number of simulation procedures have been devised to schedule trains on a single-track railway line and a treatise on some of these may be found in the article by Rudd and Storry [31]. Although simulation allows a reasonably realistic facsimile of the situation to be considered, it suffers from the drawback that conflicts are resolved uniquely as they occur and no long term effects can readily be assessed. In fact, in nearly all instances a preconceived conflict resolution scheme is employed whenever a conflict arises. This contrasts with the previously described branch and bound procedures where, if necessary, the schedules obtained by resolving all conflicts in all possible ways are generated.

In summary, although a model of a real-world situation is desirable, it seems unlikely that implicit enumeration procedures could be used to find optimal schedules. Instead, railway operators will probably have to be content with sub-optimal solutions obtained with heuristic methods or by simulation.

CHAPTER 5

ONE MACHINE SCHEDULING PROBLEMS

5.1 INTRODUCTION

The desirability of having a procedure to calculate a strong lower bound on the values of the criterion for all feasible completions of a p.f.-schedule has been briefly discussed in Chapter 2. Experience indicates that it is generally better not to expend too much computing time obtaining such bounds, *unless* they are known to be extremely good.

For the general job-shop problem with limited intermediate storage, reasonable lower bounds can be obtained by using methods based on the following general scheme:

> Given a p.f.-schedule, choose a machine and determine the time that each job, not already scheduled on it, would arrive there, assuming no further delays are incurred; that is, the machine interference constraint (condition (2.1.13)) is relaxed for all machines except the chosen one. Next, schedule these jobs on this machine so that the value of some criterion, related to the criterion for the original problem, is optimized. The solution to this single machine problem provides a lower bound for all completions of the p.f.-schedule. Finally, this process can be repeated for each machine and the maximum of the lower bounds thus obtained is the required lower bound.

The actual details of how the solution to the single machine problem is used to determine the desired lower bounds are different for each

criterion. Specific examples are provided later in the chapter. Further, the principles of the above technique have been described by Bratley, Florian and Robillard specifically for the C_{max} criterion ([4]) .

Determining an optimal schedule for a single-machine problem frequently requires the use of a branch and bound procedure. This is especially true for the problems of present interest, where the jobs are not necessarily released simultaneously. Thus, due to the obvious restrictions on computing time, one must usually be satisfied with a *lower bound* on the optimal value associated with these problems.

In the next section some notation and terminology is specified. The subsequent section provides a number of procedures for determining lower bounds for different single-machine scheduling problems with non-simultaneous release times. Although most of these procedures are known, the application of some of them for finding lower bounds is believed to be a new conception. Finally, in the last section, it is shown how these bounds can then be used to determine specific lower bounds for the general job-shop problem. One drawback of such bounding procedures is that none of them take into account the capacity restrictions on the storage bins. This is an area where further research efforts may be rewarded.

5.2 NOTATION AND TERMINOLOGY

For single-machine scheduling problems, the notation for specifying job characteristics is in accordance with that usually found in the literature. Whence, for each job j, $(j=1,\ldots,n)$ r_j , p_j , d_j and w_j represent the release time, processing time, due date and priority weighting respectively; and, as usual, c_j

represents the completion time. Further, it is convenient to adopt the four division classification of machine scheduling problems suggested by Rinnooy Kan ([30], pp. 28-29). Thus, the classification $n|1|\gamma|\delta$ refers to the problem of scheduling n jobs on one machine so that criterion δ is minimized subject to the constraints γ.

For our purposes it is necessary to always have the constraint, $r_j \geqslant 0$ (i.e. non-simultaneous release times) and sometimes it is useful to allow preemption of jobs (i.e. job splitting). This latter constraint is frequently denoted "job-spl" and will always refer to the preempt-resume discipline in which a machine resumes processing a job from where it was interrupted.

Further, it is useful to have the following definition:

a job j is *available for scheduling* at time t, if

(i) it has not been completed by time t; and

(ii) $r_j \leqslant t$. (5.2.1)

It is also convenient to clarify the concept of a "good" algorithm. Essentially,

a *good algorithm* is one which efficiently solves any instance of a problem for which it is intended, without resorting to enumerative techniques. (5.2.2)

There are some problems for which there are no known good algorithms. These problems are said to be *NP-complete*. ([23])

In the final section, when the discussion returns to the general job-shop process, the following terminology is required:

For a p.f.-schedule S and a machine i,

$J_S^i = \{j | o_{ij} \notin S\}$; i.e., the jobs not already scheduled on machine i; (5.2.3)

t_j = the time at which job $j \in J_s^i$ would arrive at machine i if it incurred no additional delays; (5.2.4)

M_{ij} = the machines on which job j has to be processed *after* it has been processed by machine i. (5.2.5)

5.3 SOME LOWER BOUNDS

Most single-machine problems for which jobs are not released simultaneously are NP-complete. One exception is the $n|1|r_j \geqslant 0|C_{max}$ problem, which is solved by processing the jobs in order of increasing release times. The criteria which are specifically investigated in this section are those listed in Chapter 2.

Two general methods for determining lower bounds for single-machine problems are proposed. The first method involves solving the single-machine problem by allowing preemption of the jobs. Here, since the optimal solution to an $n|1|r_j \geqslant 0|\delta$ problem is a feasible solution to the corresponding $n|1|r_j \geqslant 0, \text{job spl}|\delta$ problem, the value of the optimal solution to the latter problem is clearly a lower bound on the optimal value of the former. Many of the $n|1|r_j \geqslant 0, \text{job spl}|\delta$ problems can be solved with good algorithms and therefore this approach provides a fruitful source of lower bounds.

The alternative approach is to consider a corresponding $n|1|\bar{r}_j \geqslant 0|\delta$ problem for which the optimal solution is known and

$$\bar{r}_j \leqslant r_j \quad (j=1,\ldots,n) .$$

Clearly, since the release times have been relaxed, the optimal

solution to $n|1|r_j \geqslant 0|\delta$ is a feasible solution for $n|1|\bar{r}_j \geqslant 0|\delta$. Thus, whenever δ is independent of the release times, (as with all the criteria listed in Chapter 2) the value of the optimal solution to $n|1|\bar{r}_j \geqslant 0|\delta$ is a lower bound on the value of the optimal solution to $n|1|r_j \geqslant 0|\delta$, as required. When δ is dependent on the release times, the optimal solutions to the two problems are still related, but the relationship is complicated by the involvement of the release times. For example, suppose the criterion is *total delay*, in which case,

$$\delta = \sum_{j=1}^{n} \{c_j - (r_j + p_j)\} . \qquad (5.3.1)$$

Here, if for some particular data, $\bar{\delta}$ and $\delta^{\times}$ represent the optimal values of the solution for $n|1|\bar{r}_j \geqslant 0|\delta$ and $n|1|r_j \geqslant 0|\delta$ respectively, then the relationship between the optimal values is,

$$\delta^{\times} \geqslant \bar{\delta} - \sum_{j=1}^{n} (r_j - \bar{r}_j) . \qquad (5.3.2)$$

Proof

Since the optimal schedule for $n|1|r_j \geqslant 0|\delta$ is a feasible schedule for $n|1|\bar{r}_j \geqslant 0|\delta$, the delay accruing to this schedule, with respect to release times $\bar{r}_j$, is

$$\delta^{\times} + \sum_{j=1}^{n} (r_j - \bar{r}_j) ;$$

that is, $$\sum_{j=1}^{n} (c_j^{\times} - (\bar{r}_j + p_j)) = \sum_{j=1}^{n} (c_j^{\times} - (r_j + p_j)) + \sum_{j=1}^{n} (r_j - \bar{r}_j) = \delta^{\times} + \sum_{j=1}^{n} (r_j - \bar{r}_j) .$$

But the optimal solution to $n|1|\bar{r}_j \geqslant 0|\delta$ has a value $\bar{\delta}$, and therefore,

$$\delta^{\times} + \sum_{j=1}^{n} (r_j - \bar{r}_j) \geqslant \bar{\delta} .$$

The result follows.

End of Proof.

Further, when finding lower bounds by relaxing release times, one has to be careful not to relax the times too much, since the bounds may become ineffective. Thus, this approach is not recommended when the release times are well spread.

In addition to the above two general methods for determining lower bounds, there are some criteria for which a more specialized method is also available. An example of this is an approach by McMahon and Florian ([24]) for determining a lower bound for the $n|1|r_j \geq 0|L_{max}$ problem. The details of this approach are presented later.

The situations with the different criteria are now discussed separately.

5.3.1 $n|1|r_j \geq 0|\sum c_j$

For any instance of this problem, the corresponding $n|1|r_j \geq 0$, job $spl|\sum c_j$ problem can be solved using a good algorithm based on the rule: At each time t, of the jobs available for scheduling, schedule that one with the *shortest remaining processing time* (SRPT).

This rule, henceforth called the SRPT rule, is a generalization of the common shortest processing time (SPT) rule which is used to solve $n|1||\sum c_j$ problems. Although practically self-evident, the proof that this algorithm solves the $n|1|r_j \geq 0$, job $spl|\Sigma c_j$ problems is given by Horn ([14]).

Another method for determining a lower bound for the $n|1|r_j \geq 0|\sum c_j$ problem is given by Dessouky and Deogun ([8]). First it is necessary to specify a rule which they call the earliest completion time (ECT) rule. The ECT rule is:

Successively schedule the job which would finish first amongst all the unscheduled jobs. If there is a tie, choose the job which is released first.

For every schedule obtained using this rule, it is assumed that each job starts at the earliest possible time.

For a schedule S, a *block* is defined as being a set of consecutive jobs with the properties:

(i) the first job in the block is released *after* the job previously processed by the machine is finished; and

(ii) every other job in the block is released no later than the time the previously scheduled job finishes;

that is, a block consists of a sequence of contiguous jobs for which the first one commences being processed at its release time *and* after all previously processed jobs have finished. The important result proved by Dessouky and Deogun is:

Theorem 5.3.1: *A sufficient condition for the optimality of a schedule is that every block is ordered according to the ECT rule and starts with the job having the smallest release time.*

Their procedure for determining the required lower bound for an instance of the $n|1|r_j \geq 0|\sum c_j$ problem is as follows: First order the jobs according to the ECT rule; suppose the ordering is $\pi = (\pi(1),\ldots,\pi(n))$. Next, determine a schedule by setting,

$$\begin{aligned} c'_{\pi(1)} &= m_1 + p_{\pi(1)} \quad ; \quad \text{and} \\ c'_{\pi(h)} &= \max(m_h, c'_{\pi(h-1)}) + p_{\pi(h)} \quad (h = 2,\ldots,n), \end{aligned} \tag{5.3.3}$$

where

$$m_h = \min_{j \geq h} r_{\pi(j)} \quad (h=1,\ldots,n). \tag{5.3.4}$$

Clearly, if release times are prescribed by,

$$r'_{\pi(h)} = \min\{r_{\pi(h)}, \max(m_h, c'_{\pi(h-1)})\} \quad (h=1,\ldots,n), \qquad (5.3.5)$$

then, by Theorem 5.3.1, the schedule given by (5.3.3) is *optimal* for the $n|1|r'_j \geqslant 0|\sum c_j$ problem. But, by (5.3.5),

$$r'_j \leqslant r_j \quad (j=1,\ldots,n),$$

and therefore $\sum_{j=1}^{n} c'_j$ is a lower bound on the value of the optimal schedule for $n|1|r_j \geqslant 0|\sum c_j$.

Example 5.3.1

Consider the five job problem represented by the data in Table 5.3.1. Job 1 clearly would finish first amongst all the jobs and therefore, using the ECT rule, it is scheduled first. Now, since job 1 finishes at time 3, the earliest finishing times of job 2 and job 3 are 7 and 6 respectively. Thus, using the ECT rule, job 3 is scheduled next. Continuing on in this way, the following job ordering is obtained:

$$\pi = (1,3,2,5,4) .$$

The completion times obtained by applying condition (5.3.3) to this ordering are,

$$c'_1 = 0 + 3 = 3 ;$$

$$c'_3 = \max(2,3) + 2 = 5 ;$$

$$c'_2 = \max(2,5) + 4 = 9 ;$$

$$c'_5 = \max(10,9) + 2 = 12 ;$$

$$c'_4 = \max(10,12) + 4 = 16 .$$

These times prescribe the optimal solution to the problem with processing times as given by Table 5.3.1 and release times given by condition (5.3.5); i.e.,

$$r_1' = 0;\ r_3' = 3;\ r_2' = 2;\ r_5' = 10;\ r_4' = 10\ .$$

The blocks for this schedule are the sets of jobs $\{1,3,2\}$ and $\{5,4\}$, and the value associated with the schedule is

$$\sum_{j=1}^{5} c_j' = 45\ .$$

An optimal solution to the original $5|1|r_j \geqslant 0|\sum c_j$ problem has the jobs being processed by the machine in the order 1,2,3,4,5 with an associated value of 49. Further, using the SRPT rule, a lower bound of 47 is obtained.

j	r_j	p_j
1	0	3
2	2	4
3	4	2
4	10	4
5	11	2

Table 5.3.1: Data for Example 5.3.1

To see which of the job-splitting technique and Dessouky and Deogun's method gives the better bounds, both procedures have been programmed and run with the same random data. The release times and the processing times were chosen from a uniform distribution on $[0,r_{max}]$ and $[1,p_{max}]$ respectively. For each configuration of r_{max}, p_{max} and n, the bounds for five problems have been determined. Some tests with different combinations of r_{max} and p_{max} have shown that each method takes about 0.060 seconds of central processor time to run for problems with 40 jobs. No further attempt has been made to compare the running times of the two procedures nor to compare the bounds against the values of the optimal solutions.

Surprisingly, in *every* situation, the bound obtained using job-splitting (denoted LB_{JS}) was at least as good as the bound (LB_{DD}) obtained using Dessouky and Deogun's method. As an indication of the difference between the bounds, the measure,

$$\left(\frac{1}{5} \sum \frac{LB_{JS} - LB_{DD}}{LB_{DD}}\right) \times 100\%$$

is used, where the summation is over all the five problems for each set of r_{max}, p_{max} and n values. This measure can be interpreted as the *average relative percentage improvement* of LB_{JS} on LB_{DD}. Using relative values in this way enables a direct comparison of all configurations to be made. The results are presented in Table 5.3.2.

Although no absolute conclusions can be drawn from these results, there are a number of general trends which are apparent. For example, it seems likely that,

1. the number of problems for which LB_{JS} equals LB_{DD} decreases as the number of jobs increases *and* as the job release times become more widespread relative to the processing times; and
2. as long as r_{max} is not too much larger than p_{max}, the relative percentage difference between LB_{JS} and LB_{DD} decreases as n increases.

Further, one might expect that problems with widespread release times and large processing times are comparable to problems with closer release times and smaller processing times. Using the notation,

$$(r_{max}, p_{max}, n)$$

to represent a configuration and the symbol $\sim$ to denote a "rough" equivalence of values (i.e. within an order of magnitude), this

r_{max}	No. of jobs n	p_{max} = 10		p_{max} = 20		p_{max} = 50	
		No. of Problems in which $LB_{JS}=LB_{DD}$	$\sum \frac{100(LB_{JS}-LB_{DD})}{5\ LB_{DD}}$ %	No. of Problems in which $LB_{JS}=LB_{DD}$	$\sum \frac{100(LB_{JS}-LB_{DD})}{5\ LB_{DD}}$ %	No. of Problems in which $LB_{JS}=LB_{DD}$	$\sum \frac{100(LB_{JS}-LB_{DD})}{5\ LB_{DD}}$ %
10	5	4	4.23	1	5.17	2	2.96
	10	3	1.29	1	1.39	0	1.03
	20	1	1.14	2	0.52	1	0.40
	40	1	0.25	2	0.07	0	0.10
20	5	1	3.87	2	10.14	1	3.52
	10	1	5.71	0	3.53	0	4.17
	20	0	2.53	1	1.41	0	1.14
	40	2	0.27	0	0.55	1	0.15
40	5	2	1.95	3	2.60	1	2.79
	10	0	11.19	1	4.88	0	3.51
	20	0	3.84	1	1.04	1	1.18
	40	0	0.64	0	0.19	0	0.39
100	5	0	1.36	2	1.71	1	13.10
	10	0	1.41	0	6.06	1	4.81
	20	0	3.56	0	4.64	0	2.22
	40	0	2.29	0	1.62	0	0.82

Table 5.3.2

conjecture is borne out by Table 5.3.2; that is, for $n = 5, 10, 20$ and 40,

$$(100,50,n) \sim (40,20,n) \sim (20,10,n) .$$

Finally, since both procedures appear to take about the same time to run, it seems likely that the job-splitting technique should always be used in preference to Dessouky and Deogun's method, when finding lower bounds for $n|1|r_j \geqslant 0|\sum c_j$ problems.

5.3.2 $n|1|r_j \geqslant 0|\sum w_j c_j$

Any instance of the $n|1||\sum w_j c_j$ problem is solved by scheduling the jobs in order of non-decreasing p_j/w_j ratios. The obvious extension to the $n|1|r_j \geqslant 0$, job spl $|\sum w_j c_j$ problem is the following:

> At any time t, schedule the available job for which the ratio of *remaining* execution requirement to priority weighting is smallest.

Unfortunately this rule, henceforth called the SRPT/w rule, does not always produce optimal schedules. For example, consider the problem represented by the data in Table 5.3.3. Using the SRPT/w rule gives the schedule (non-preemptive as it happens) shown in figure 5.3.1(a). For this schedule, $\sum w_j c_j = 380$, which is 12 units more than for schedule (b) in figure 5.3.1.

j	r_j	p_j	w_j
1	0	2	5
2	0	1	2
3	1	1	4
4	2	2	9
5	3	1	8
6	4	2	17
7	5	1	16

Table 5.3.3

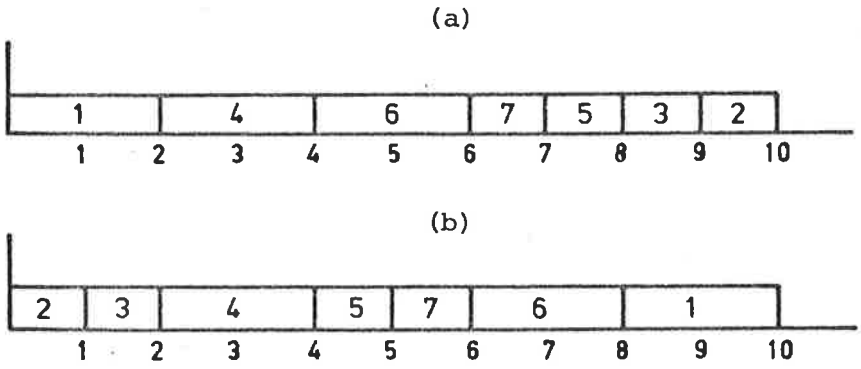


Figure 5.3.1

In fact the $n|1|r_j \geqslant 0$, job spl $|\ \sum w_j c_j$ problem has been shown in [19] to be NP-complete. But, it is not necessary to discard the SRPT/w rule, since it can be used ([19]) to solve any instance of the $n|1|r_j \geqslant 0$, job spl $|\ \sum w_j c_j$ problem for which the weights are *agreeable*; i.e.

$$p_i > p_j \Rightarrow w_i \leqslant w_j \ . \tag{5.3.6}$$

Therefore, if the weightings are *reduced*, where necessary, to $\bar{w}_j$ so that the $\bar{w}_j$ $(j=1,\ldots,n)$ are agreeable, then SRPT/w can be used to solve the $n|1|r_j \geqslant 0$, job spl $|\ \sum \bar{w}_j c_j$ problem. But the value of $\sum \bar{w}_j c_j$ obtained in this way is a lower bound on the value of the optimal solution for $n|1|r_j \geqslant 0|\sum \bar{w}_j c_j$, which in turn provides the required lower bound for $n|1|r_j \geqslant 0|\sum w_j c_j$. This last implication follows from the fact that for any schedule with completion times c_j,

$$\sum \bar{w}_j c_j \leqslant \sum w_j c_j \ .$$

Example 5.3.2

To determine a lower bound for the problem prescribed by the data in Table 5.3.3, it is first necessary to reduce the weightings

so that they are agreeable. Now, since p_1, p_4 and p_6 are all greater than p_2, then, to satisfy (5.3.6), w_1, w_4 and w_6 have to be reduced to the value of w_2, viz. 2. The new problem, given by Table 5.3.4, can now be solved using the SRPT/w rule. This gives a schedule with a value of 184, which is a lower bound on the optimal value for the original problem.

j	r_j	p_j	$\bar{w}_j$
1	0	2	2
2	0	1	2
3	1	1	4
4	2	2	2
5	3	1	8
6	4	2	2
7	5	1	16

Table 5.3.4: The new data

When applying the SRPT/w rule to the $n|1|r_j \geqslant 0, \text{job spl}|\sum w_j c_j$ problems, it is only really necessary for the weightings of the jobs available for scheduling at each time point to be agreeable. Therefore, instead of making all weights agreeable from the outset, a better bound can be obtained if the weights of *available* jobs are made agreeable at the time each *new* job becomes available for scheduling. The final weights of the jobs are no smaller than if they had been made agreeable from the outset.

Example 5.3.3

Consider again the data in Table 5.3.3. At time 0, both jobs 1 and 2 are available for scheduling, and so w_1 has to be reduced to the value of w_2, viz. 2. Using the SRPT/w rule, jobs 2 and 3 are completely scheduled before job 4 becomes available at time 2. Further, when job 5 becomes available at time 3, only one unit of

job 4 still has to be scheduled and therefore the weights at this time point are still agreeable. Continuing on in this manner, the only other change is that w_6 has to be reduced to 8 to make it agreeable with w_5. The schedule thus obtained is actually the one shown in figure 5.3.1(b); but, because of the alterations to the weights, it now has a value of 266 associated with it. This quantity is a superior lower bound on the value of the optimal schedule for the original problem.

Finally, it seems that both the above methods are innovations for finding lower bounds for the $n|1|r_j \geq 0|\sum w_j c_j$ problems.

An alternative method of finding a lower bound for the above problem is to reduce all release times to the minimum of them and then solve the corresponding $n|1||\sum w_j c_j$ problem. The lower bound obtained in this way is clearly not very satisfactory when release times are wide-spread. Applying this method to the data given in Table 5.3.3 provides a lower bound of 246 for the given problem.

5.3.3 $n|1|r_j \geq 0|\sum U_j$

A current open question in the area of complexity results for machine scheduling problems is whether or not $n|1|r_j \geq 0$, job spl $|\sum U_j$ is NP-complete (see [12]). Accordingly, there are no known good algorithms for solving this problem and so there is little future in investigating job splitting techniques for determining lower bounds for $n|1|r_j \geq 0|\sum U_j$ problems.

Now, although $n|1|r_j \geq 0|\sum U_j$ is known to be NP-complete for arbitrary release times, any instance is solvable with a good algorithm whenever,

$$r_i < r_j \Rightarrow d_i \leq d_j \quad . \tag{5.3.7}$$

An algorithm used to solve such problems has been presented by Kise, Ibaraki and Mine ([17]). Assuming the n jobs are indexed so that $r_1 \leqslant r_2 \leqslant \ldots \leqslant r_n$ and $d_1 \leqslant d_2 \leqslant \ldots \leqslant d_n$, their algorithm is essentially the following:

> In order of the indices, successively schedule jobs at the earliest possible times until one is found to be tardy. Then, discard that job whose removal leads to a partial schedule with the smallest finishing time. Continue scheduling and discarding jobs until all of them have been considered.

The number of discarded jobs is the minimum possible number of tardy jobs. Discarding a job means that it is processed by the machine after all the non-tardy jobs have been finished.

Example 5.3.4

Consider the problem represented by the data in Table 5.3.5. Condition (5.3.7) is satisfied and therefore the above algorithm can be applied.

j	r_j	p_j	d_j
1	0	4	7
2	2	4	8
3	4	3	9
4	5	2	10
5	6	3	11

Table 5.3.5: Data for Example 5.3.4

Now, job 1 and job 2 can be scheduled successively with neither being tardy; but job 3 cannot be added to this schedule without it being tardy. The three partial schedules to be considered at this stage can be represented by the job sequences,

1,2 ; 1,3 and 2,3 .

The finishing times associated with these schedules are 8, 7 and 9 respectively, and therefore 1,3 is retained; i.e. job 2 is discarded. Job 4 can be added to this schedule successfully but job 5 cannot be added to the resulting schedule without being tardy. Of the four sequences,

1,3,4 ; 1,3,5 ; 1,4,5 and 3,4,5,

sequence 1,3,4 corresponds to the schedule with the earliest finishing time and therefore job 5 is discarded. Thus, for this problem, the minimum number of tardy jobs is 2.

Returning to the general $n|1|r_j \geqslant 0|\sum U_j$ problem, suppose the jobs are indexed according to increasing due dates with ties resolved by choosing first, the job with the smallest release time. Then, if the release times r_j $(j=1,\ldots,n)$ are changed to values $\bar{r}_j$ in the following way:

$$\bar{r}_n = r_n \ ; \ \text{and}$$
$$\bar{r}_j = \min(r_j, \bar{r}_{j+1}) \ (j=n-1,\ldots,1) \ , \qquad (5.3.8)$$

condition (5.3.7) is never violated and $n|1|\bar{r}_j \geqslant 0|\sum U_j$ can be solved using Kise, Ibaraki and Mine's algorithm. But, from (5.3.8),

$$\bar{r}_j \leqslant r_j \ (j=1,\ldots,n) \ ,$$

and therefore the value of the optimal solution to $n|1|\bar{r}_j \geqslant 0|\sum U_j$ provides a lower bound for $n|1|r_j \geqslant 0|\sum U_j$, as required. It is believed that Kise, Ibaraki and Mine's algorithm has not previously been used in this way.

5.3.4 $n|1|r_j \geqslant 0|L_{max}$

For any instance of this problem, the corresponding $n|1|r_j \geqslant 0$,

job $spl|L_{max}$ problem can be solved using an algorithm based on the rule:

> Of the jobs which are available at each release time or the time at which a job is completed, schedule the one with the earliest due date (EDD).

This rule is a generalization of the EDD rule used to solve $n|1||L_{max}$. The proof that the above algorithm does indeed solve $n|1|r_j \geq 0$, job $spl|L_{max}$ problems is again given in Horn [14]. Thus, using job-splitting techniques, a lower bound for $n|1|r_j \geq 0|L_{max}$ can be obtained.

Alternatively, a lower bound can be obtained using the previously mentioned approach of McMahon and Florian [24]. Their method is as follows:

First, an obvious lower bound is

$$LB = \max_j (r_j + p_j - d_j) . \qquad (5.3.9)$$

Next, it is necessary to schedule the jobs using the following heuristic algorithm proposed by Schrage, where $\bar{S}$ is the set of unscheduled jobs:

1. Set $t = 0$
2. Is there at least one job $j \in \bar{S}$ such that $r_j \leq t$. If so, go to Step 4.
3. Set $t = \min_{j \in \bar{S}} r_j$.
4. Among all jobs $j \in \bar{S}$ such that $r_j \leq t$, choose that job j that has the smallest due date; break ties by selecting the job with the longest processing time.
5. Schedule the chosen job next and set t to $t + p_j$.

6. If $\bar{S} \neq \phi$, go to 2. Otherwise, stop.

The sequence of jobs $\pi(1),\ldots,\pi(n)$ produced by this heuristic can be decomposed into *blocks* in this same way as they could in sub-section 5.3.1. The following definitions are required: For each job j in block $\{\pi(g),\ldots,\pi(h)\}$,

1. $P_j = \{i \mid c_{\pi(g)} \leqslant c_i \leqslant c_j\}$; (5.3.10)

2. $d_j^* = \max\{d_i \mid i \in P_j\}$; (5.3.11)

3. $Q_j = \{i \in P_j \mid d_i = d_j^*\}$. (5.3.12)

With these definitions, it can be shown ([20]) that for,

$$LB_j = \begin{cases} c_j - d_j^* & \text{if } j \in Q_j \text{ ;} \\ c_j + 1 - d_j^* & \text{if } j \notin Q_j \text{ ,} \end{cases} \tag{5.3.13}$$

$$\max LB_j \tag{5.3.14}$$

is a lower bound for $n|1|r_j \geqslant 0|L_{max}$.

Example 5.3.5

Consider the problem specified by the data in Table 5.3.6. Scheduling the jobs using Schrage's heuristic gives the schedule represented by the Gantt chart in figure 5.3.2. The blocks (B_i say) associated with this schedule are,

$$B_1 = \{1,3,2\} \quad \text{and} \quad B_2 = \{4,5\} .$$

The job completion times, c_j, for this schedule and the values LB_j are presented in Table 5.3.7 together with quantities determined by (5.3.10) through (5.3.12). Clearly, from this table,

$$\max_j LB_j = 3 ,$$

which is a lower bound on the value of the optimal schedule for the prescribed $5|1|r_j \geqslant 0|L_{max}$ problem. Since the schedule

j	r_j	p_j	d_j
1	0	3	5
2	1	3	5
3	2	2	4
4	9	2	14
5	10	3	13

Table 5.3.6: Data for Example 5.3.5

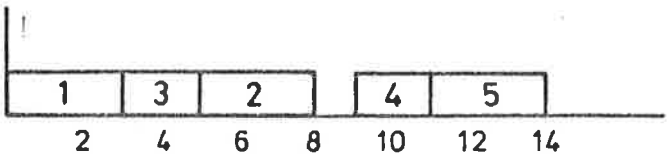


Figure 5.3.2

j	c_j	P_j	d_j^*	Q_j	LB_j
1	3	{1}	5	{1}	-2
2	8	{1,3,2}	5	{1,2}	3
3	5	{1,3}	5	{1}	1
4	11	{4}	14	{4}	-3
5	14	{4,5}	14	{4}	1

Table 5.3.7

shown in figure 5.3.2 has a value of L_{max} equal to 3, this lower bound is achieved. Thus, the schedule determined by Schrage's algorithm is optimal in this instance. For completeness, as both (5.3.9) and (5.3.14) give a lower bound on $n|1|r_j \geqslant 0|L_{max}$, a better bound is given by the maximum of these two values.

Finally, there is no need to determine lower bounds for $n|1|r_j \geqslant 0|C_{max}$ problems since, as noted earlier, these problems can be solved directly by scheduling the jobs in order of increasing release times.

5.4 LOWER BOUNDS FOR THE GENERAL PROBLEM

The general method for determining lower bounds on the values of all completions of a p.f.-schedule S has been outlined at the commencement of this chapter. In this section the exact way these bounds are obtained for different criteria is described.

Now, if for an arbitrary feasible completion of S, the start times and finish times of operation O_{ij} are denoted by a_{ij} and b_{ij} respectively and if c_j is the time at which job j is completed, then, for any machine i $(i=1,\ldots,m)$,

$$c_j \geqslant b_{ij} + \sum_{h \in M_{ij}} d_{hj} \quad . \tag{5.4.1}$$

Recalling the definition ((5.2.5)) of M_{ij}, this inequality follows from the fact that the right hand side of (5.4.1) is the time job j would finish if it incurred no additional delays after leaving machine i.

5.4.1 Criterion $\sum c_j$

For any completion of S, the value of this criterion is $\sum_{j=1}^{n} c_j$; which, by (5.4.1) satisfies the inequality,

$$\sum_{j=1}^{n} c_j \geqslant \sum_{j=1}^{n} b_{ij} + \sum_{j=1}^{n} \sum_{h \in M_{ij}} d_{hj} \quad (i=1,\ldots,m) \quad . \tag{5.4.2}$$

For machine i $(i=1,\ldots,m)$ and for any subset J of n_1 jobs, consider the corresponding one-machine problem with release

times a_{ij} and processing times d_{ij}, for which the total of the completion times is to be minimized. The value of the optimal solution to this $n|1|a_{ij} \geqslant 0|\sum c_j$ problem is $\sum_{j \in J} b_{ij}$, since every job can commence at its release time. Therefore, (5.4.2) can be rewritten as,

$$\sum_{j=1}^{n} c_j \geqslant \sum_{j \in J} b_{ij} + \sum_{j=1}^{n} \sum_{h \in M_{ij}} d_{hj} + \text{the value of the optimal solution to } n_1|1|a_{ij} \geqslant 0|\sum c_j \ . \qquad (5.4.3)$$

Now, recalling the definitions (5.2.3) and (5.2.4) for J_S^i and t_j respectively, for job $j \in J_S^i$ and machine i, let

$$\bar{r}_j = \max(t_j , \max_{h \notin J_S^i} b_{ih}) \ ; \qquad (5.4.4)$$

that is, from the situation described by S, $\bar{r}_j$ is the earliest time j could become available to machine i so that it would commence at least after those operations (in S) already executed by machine i have finished. But clearly,

$$\bar{r}_j \leqslant a_{ij} \ ,$$

and therefore, if J_S^i contains n_1 jobs to be scheduled on machine i, the optimal solution to $n_1|1|a_{ij} \geqslant 0|\sum c_j$ is a feasible solution to $n|1|\bar{r}_j \geqslant 0|\sum c_j$, where the problems are identical except for release times. Thus, the value δ_i^x, say, of the optimal solution to the $n_1|1|\bar{r}_j \geqslant 0|\sum c_j$ problem is a lower bound on the value of the optimal solution to $n_1|1|a_{ij} \geqslant 0|\sum c_j$. Whence, from (5.4.3),

$$\sum_{j=1}^{n} c_j \geqslant \sum_{j \notin J_S^i} b_{ij} + \sum_{j=1}^{n} \sum_{h \in M_{ij}} d_{hj} + \delta_i^x \ . \qquad (5.4.5)$$

Now, each term on the right hand side of inequality (5.4.5) is

dependent only on S and i and *not* on the actual completion of S. Therefore,

$$\sum_{j \notin J_S^i} b_{ij} + \sum_{j=1}^{n} \sum_{h \in M_{ij}} d_{hj} + \delta_i^{\times}$$

is a lower bound on the criterion values for *all* completions of S. Further, although $\delta_i^{\times}$ cannot be determined with a good algorithm, it has been shown in the previous section that a lower bound δ_i^{LB}, say, can be found for this value using job-splitting techniques or relaxing release times. Moreover, the choice for machine i is arbitrary and so a lower bound which is readily calculated is,

$$\max_i \{ \sum_{j \notin J_S^i} b_{ij} + \sum_{j=1}^{n} \sum_{h \in M_{ij}} d_{hj} + \delta_i^{LB} \} \quad . \tag{5.4.6}$$

5.4.2 Criterion $\sum w_j c_j$

Suppose $\delta_i^{\times}$ is the value of the optimal schedule for $n_1|1|\bar{r}_j \geq 0|\sum w_j c_j$, where $\bar{r}_j$, given by (5.4.4), is the release time for job $j \in J_S^i$. Then, using a derivation analogous to that for criterion $\sum c_j$, for *any* completion of a p.f.-schedule S,

$$\sum_{j=1}^{n} w_j c_j \geq \sum_{j \notin J_S^i} w_j b_{ij} + \sum_{j=1}^{n} \sum_{h \in M_{ij}} w_j d_{hj} + \delta_i^{\times} \quad . \tag{5.4.7}$$

Continuing the analogy, methods are available for determining a lower bound δ_i^{LB} on $\delta_i^{\times}$. Consequently, a lower bound on the value of the criterion, $\sum w_j c_j$, for every feasible completion of S is:

$$\max_i \{ \sum_{j \notin J_S^i} w_j b_{ij} + \sum_{j=1}^{n} \sum_{h \in M_{ij}} w_j d_{hj} + \delta_i^{LB} \} \quad . \tag{5.4.8}$$

5.4.3 Criterion $\sum U_j$

For any feasible completion of p.f.-schedule S,

$$U_j = \begin{cases} 1 & \text{if } c_j > d_j \ ; \\ 0 & \text{otherwise .} \end{cases}$$

Therefore, defining for machine i,

$$U_j' = \begin{cases} 1 & \text{if } b_{ij} + \sum_{h \in M_{ij}} d_{ij} > d_j \ ; \\ 0 & \text{otherwise ,} \end{cases} \tag{5.4.9}$$

the inequality (5.4.1) gives,

$$U_j \geqslant U_j' \qquad (j=1,\ldots,n) \ ,$$

and therefore ,

$$\sum_{j=1}^{n} U_j \geqslant \sum_{j=1}^{n} U_j' \ . \tag{5.4.10}$$

Now, for machine i and any set J of n_1 jobs, consider the associated single-machine problem with release times a_{ij}, processing times d_{ij} and due dates $d_j - \sum_{h \in M_{ij}} d_{hj}$, for which the number of tardy jobs is to be minimized. Since jobs can commence at their release times, the value of the optimal schedule for this $n_1|1|a_{ij} \geqslant 0|\sum U_j$ problem is,

$$\left|\{j \mid b_{ij} > d_j - \sum_{h \in M_{ij}} d_{hj}\}\right| \ ,$$

which, by (5.4.9) equals

$$\sum_{j \in J} U_j' \ .$$

Thus, (5.4.10) gives

$$\sum_{j=1}^{n} U_j \geqslant \sum_{j \notin J} U_j' + \sum_{j \in J} U_j' \ . \tag{5.4.11}$$

In an analogous way to that for the previous criteria, if the $\bar{r}_j$ $(j \in J_S^i)$ are given by (5.4.4) and the jobs in J_S^i are yet to be scheduled on machine i, then the value of the optimal schedule for $n_1|1|\bar{r}_j \geqslant 0|\sum U_j$ with due dates $d_j - \sum_{h \in M_{ij}} d_{hj}$ is a lower bound on $\sum_{j \in J_S^i} U_j'$. Letting this optimal value be $\delta_i^{\times}$, (5.4.11) becomes

$$\sum_{j=1}^{n} U_j \geqslant \sum_{j \notin J_S^i} U_j' + \delta_i^{\times} \quad . \tag{5.4.12}$$

The right hand side of this inequality is independent of the actual completion of S and therefore is a lower bound for every completion.

Again, it is generally more efficacious to obtain a lower bound on $\delta_i^{\times}$, using one of the methods suggested in the previous section, rather than finding $\delta_i^{\times}$ itself. Letting this lower bound be δ_i^{LB}, the bound

$$\sum U_j' + \delta_i^{LB}$$

is readily obtainable. Finally, since this bound is determined for arbitrary i, a better lower bound is

$$\max_i \{ \sum_{j \notin J_S^i} U_j' + \delta_i^{LB} \} \quad . \tag{5.4.13}$$

5.4.4 Criterion L_{max}

Suppose the jobs are partitioned into two sets J and $\bar{J}$. Then, from (5.4.1), for any completion of the p.f.-schedule S and for any machine i,

$$\max_j (c_j - d_j) \geqslant \max\{ \max_{j \in \bar{J}} (b_{ij} + \sum_{h \in M_{ij}} d_{hj} - d_j), \max_{j \in J} (b_{ij} + \sum_{h \in M_{ij}} d_{hj} - d_j) \} \tag{5.4.14}$$

For jobs in J, the optimal solution to the $n_1|1|a_{ij} \geqslant 0|L_{max}$ problem with due dates $d_j - \sum_{h \in M_{ij}} d_{hj}$ and processing times d_{ij} has a value,

$$\max_{j \in J} (b_{ij} + \sum_{h \in M_{ij}} d_{hj} - d_j) .$$

As for previous criteria, this follows since jobs can commence being processed by the machine *at* their release times. Now, in the usual way, if $\bar{r}_j$ is given by (5.4.4) ($j \in J_S^i$) and the jobs in J_S^i are to be scheduled on machine i, then the value, $\delta_i^{\times}$, of the optimal solution to the $n_1|1|\bar{r}_j \geqslant 0|L_{max}$ problem with due dates $d_j - \sum_{h \in M_{ij}} d_{hj}$ is a lower bound on

$$\max_{j \in J_S^i} (b_{ij} + \sum_{h \in M_{ij}} d_{hj} - d_j) .$$

Further, if δ_i^{LB} is a lower bound for $\delta_i^{\times}$, then (5.4.14) gives,

$$\max_j (c_j - d_j) \geqslant \max\{\max_{j \notin J_S^i} (b_{ij} + \sum_{h \in M_{ij}} d_{hj} - d_j), \delta_i^{LB}\}. \qquad (5.4.15)$$

But, since the choice of machine i is arbitrary and since the right hand side of (5.4.15) is independent of the *actual* completion of S,

$$\max_i [\max\{\max_{j \notin J_S^i} (b_{ij} + \sum_{h \in M_{ij}} d_{hj} - d_j), \delta_i^{LB}\}] \qquad (5.4.16)$$

is a lower bound on the values for all completions of S.

5.4.5 Criterion C_{max}

The easiest way to determine the required lower bound for this criterion is to realise that C_{max} is just a special case of L_{max} with due dates all equal to zero. Therefore, for any p.f.-schedule S, a lower bound on the values of all completions of this schedule is given by,

$$\max_i [\max\{\max_{j \notin J_S^i} (b_{ij} + \sum_{h \in M_{ij}} d_{hj}), \delta_i^{LB}\}] , \qquad (5.4.17)$$

where δ_i^{LB} is a lower bound on the value of the optimal solution to the $n_1|1|\bar{r}_j \geq 0|L_{max}$ problem with processing times d_{ij} and due dates $-\sum_{h \in M_{ij}} d_{hj}$.

5.4.6 Concluding Remarks

For all the results in the thesis obtained using branch and bound procedures, the bounding routine employed was that described in Chapter 2; that is, the one for which projected job completion times are used. Now, although the bounding methods described in this section would produce better bounds than the bounding routine actually used, their implementation requires many more calculations and comparisons. Thus, since bounds have to be calculated numerous times during the course of an algorithm, it has been deemed prudent to use the quickest method in preference to the methods producing superior bounds. This trade-off between computing time and the utility of the bound will always favour the quicker methods until very potent bounding routines are developed. Such routines will necessarily consider the capacities of the bins and may also employ some of the results presented in this chapter.

Finally, if one so desired, any of the bounding methods described in this section could readily be included in the branch and bound procedures described earlier.

CHAPTER 6

DISCUSSION

The purpose of this thesis has been to develop a general framework for solving problems which can be cast into the form of a job-shop process with intermediate storage facilities of limited capacity. Compared to the more usually studied situations with unrestricted storage, these problems have been shown to require complicated models whose attributes have to be described in very explicit terms.

A major feature of the thesis is the realization that the general process can be represented by two fundamentally different models; one corresponds to jobs having to visit each of their assigned storage locations, the other allows jobs to bypass these locations whenever possible. The key to these models is whether the waiting intervals are treated as half-open or closed. Further, when these intervals are closed, it has been seen that a critical value δ must be prescribed so that one can decide when such intervals interact (as opposed to overlap) with each other. For half-open waiting intervals, the concept of "strings of operations" has had to be defined to enable the start times of consecutive operations for the same job to be reduced simultaneously.

Corresponding to each of the above models, it is necessary to have a very precise definition of a semi-active schedule. Moreover, both definitions allow for the possibility of there being more than one semi-active schedule per feasible ordering of the jobs on the machines, in marked contrast to the processes with no

restrictions on storage. Fortunately, the total number of semi-active schedules is still finite and therefore an optimal schedule can be obtained by enumerating them all and choosing one for which the value of the criterion is smallest.

The solution methods presented in the thesis have, by necessity, been based on implicit enumeration techniques, more specifically, branch and bound procedures. For different specializations of the general process, it is desirable at each stage of the enumeration procedure to be able to restrict the set of operations available for scheduling in such a way that all semi-active schedules can be generated without duplication. This has been accomplished for the specialization in Chapter 3,where the jobs all follow the same route through the job-shop and also in Chapter 4, where a train scheduling problem has been modelled in a way which corresponds to jobs being able to take one of two routes through the shop. Such solution algorithms are the "best-possible" in the sense that they enable the entire set of semi-active schedules to be explored without duplication. Unless a set of schedules can be found which is always a subset of the semi-active schedules and is known to still contain an optimal schedule, then no improvement on these procedures is possible.

Methods have also been described for determining a lower bound on the values of the criterion for all completions of a partial feasible schedule. These methods require some results and procedures pertaining to the scheduling of jobs on a single machine. In many instances, these procedures have been implemented in novel ways so that the appropriate lower bounds can be found.

Finally, the research performed for the thesis indicates some areas where further investigation may produce results. These areas

include the specialization of the general models to other real-world problems, the development of reliable heuristics for solving large problems and the construction of lower bounding routines which take into account the capacity restrictions on the storage bins.

INDEX OF NOTATION AND TERMINOLOGY

BIBLIOGRAPHY

1. ANDERSON, I. (1974). *A First Course in Combinatorial Mathematics*. Clarendon Press, Oxford.

2. BAKER, K.R. (1974). *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York.

3. BONNEY, M.C. and GUNDRY, S.W. (1976). *Solutions to the Constrained Flowshop Sequencing Problem*. Operational Research Quarterly, Vol. 27, 2, pp. 869-883.

4. BRATLEY, P., FLORIAN, M. and ROBILLARD, P. (1973). *On Sequencing with Earliest Starts and Due Dates with Application to Computing Bounds for the $(n|m|G|F_{max})$ Problem*. Naval Research Logistics Quarterly, Vol. 20, 1, pp. 57-67.

5. CHERNIAVSKY, A.L. (1972). *A Program for Timetable Compilation by a Look-Ahead Method*. Artificial Intelligence, Vol. 3, pp. 61-76.

6. CONWAY, R.W., MAXWELL, W.L. and MILLER, L.W. (1967). *Theory of Scheduling*. Addison Wesley, Reading, Mass.

7. DANNENBRING, D.G. (1977). *An Evaluation of Flow Shop Sequencing Heuristics*. Management Science, Vol. 23, 11, pp. 1174-1182.

8. DESSOUKY, M.I. and DEOGUN, J.S. (1979). *Sequencing Jobs with Unequal Ready Times to Minimize Mean Flow Time*. Presented at TIMS-ORSA Meeting, New Orleans, La., May 1979.

9. DUTTA, S.K. and CUNNINGHAM, A.A. (1975). *Sequencing Two-Machine Flow-Shops with Finite Intermediate Storage*. Management Science, Vol. 21, 9, pp. 989-996.

10. ELMAGHRABY, S.E. (Ed.) (1973). *Symposium on the Theory of Scheduling and its Applications*. Lecture Notes in Economics and Mathematical Systems No. 86, Springer-Verlag, Berlin.

11. GIFFLER, B. and THOMPSON, G.L. (1960). *Algorithms for Solving Production-Scheduling Problems.* Operations Research, Vol. 8, 4, pp. 487-503.

12. GRAHAM, R.L., LAWLER, E.L., LENSTRA, J.K. and RINNOOY KAN, A.H.G. (1977). *Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey.* Proceedings of "Discrete Optimization 1977" Vancouver, August 1977. (To be published).

13. GUPTA, J.N.D. (1976). *Optimal Flowshop Schedules with no Intermediate Storage Space.* Naval Research Logistics Quarterly, Vol. 23, 2, pp. 235-243.

14. HORN, W.A. (1974). *Some Simple Scheduling Algorithms.* Naval Research Logistics Quarterly, Vol. 21, 1, pp. 177-185.

15. JONES, J.C.M. and WALKER, A.E. (1973). *The Application of Models of Single Railway Track Operation to Evaluate Upgrading Alternatives.* Rail International, 4th Year, No. 7, pp. 787-801.

16. KARP, R.M. (1975). *On the Computational Complexity of Combinatorial Problems.* Networks, Vol. 5, pp. 45-68.

17. KISE, H., IBARAKI, T. and MINE, H. (1978). *A Solvable Case of the One-Machine Scheduling Problem with Ready and Due Times.* Operations Research, Vol. 26, 1, pp. 121-126.

18. KNUTH, D.E. (1973). *The Art of Computer Programming Volume 3 - Sorting and Searching.* Addison Wesley, Reading, Mass.

19. LABETOULLE, J., LAWLER, E.L., LENSTRA, J.K. and RINNOOY KAN, A.H.G. (1978). *Preemptive Scheduling of Uniform Machines Subject to Release Dates.* Proceedings of the Summer School in Combinatorial Optimization, Sogesta, Urbino, Italy, July 1978. (To be published).

20. LAGEWEG, B.J., LENSTRA, J.K. and RINNOOY KAN, A.H.G. (1976). *Minimizing Maximum Lateness on One Machine: Computational Experience and Some Applications*. Statistica Neerlandica, Vol. 30, 1, pp. 25-41.

21. LAWLER, E.L. (1976). *Sequencing to Minimize the Weighted Number of Tardy Jobs*. Revue Francaise D'Automatique Informatique Recherche Operationnelle, Vol. 10, 5, Supplement pp. 27-33.

22. LAWLER, E.L. and WOOD, D.E. (1966). *Branch and Bound Methods: A Survey*. Operations Research, Vol. 14, 4, pp. 699-719.

23. LENSTRA, J.K., RINNOOY KAN, A.H.G. and BRUCKER, P. (1977). *Complexity of Machine Scheduling Problems*. Annals of Discrete Mathematics, Vol. 1, pp. 343-362.

24. McMAHON, G. and FLORIAN, M. (1975). *On Scheduling with Ready Times and Due Dates to Minimize Maximum Lateness*. Operations Research, Vol. 23, 3, pp. 475-482.

25. MOORE, J.M. (1968). *An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs*. Management Science, Vol. 15, 1, pp. 102-109.

26. NIJENHUIS, A. and WILF, H.S. (1978). *Combinatorial Algorithms for Computers and Calculators* (2nd Edition). Academic Press, New York.

27. OTWAY, N.J. and SALZBORN, F.J.M. (1979). *Generating Feasible Schedules for a Single Track Railway Line*. To appear.

28. REDDI, S.S. and RAMAMOORTHY, C.M. (1972). *On the Flowshop Sequencing Problem with No Wait in Process*. Operational Research Quarterly, Vol. 23, 3, pp. 323-331.

29. REDDI, S.S. and RAMAMOORTHY, C.M. (1973). *A Scheduling Problem.* Operational Research Quarterly, Vol. 24, 3, pp. 441-446.

30. RINNOOY KAN, A.H.G. (1976). *Machine Scheduling Problems: Classification, Complexity and Computations.* Nijhoff, The Hague.

31. RUDD, D.A. and STORRY, A.J. (1976). *Single Track Railway Simulation - New Models and Old.* Rail International, 7th Year, No. 6, pp. 335-342.

32. VAN DEMAN, J.M. and BAKER, K.R. (1974). *Minimizing Mean Flowtime in the Flow Shop with No Intermediate Queues.* American Institute of Industrial Engineers, Transactions, Vol. 6, 1, pp. 28-34.

33. WISMER, D.A. (1972). *Solutions of the Flowshop Scheduling Problem with No Intermediate Queues.* Operations Research, Vol. 20, 3, pp. 689-697.