

BA

**stichting
mathematisch
centrum**

**M
MC**

AFDELING MATHEMATISCHE BESLISKUNDE BW 27/73 AUGUST
IN SAMENWERKING MET
HET INTERUNIVERSITAIR INSTITUUT BEDRIJFSKUNDE
DELFT/ROTTERDAM

A.H.G. RINNOOY KAN
THE MACHINE SCHEDULING PROBLEM

BA

2e boerhaavestraat 49 amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
 AMSTERDAM

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

ERRATA

- Page 32, last line : for "T" read " τ "
- Page 33, 2nd line : for ">" read " \gg "
- Page 39, 12th line : for " $f(1,1) = 17$ " read " $f(0,1) = 14$ "
- Page 39, 13th line : for " $f(1,2) = 20$ " read " $f(0,2) = 17$ "
- Page 70, 2nd line : for "increasing" read "decreasing"
- Page 81, 10th line : for " $p_{i_{\ell-1}}$ " read " $p_{i_{k-1}}$ "
- Page 81, last line : for " $s_k(t)$ " read " $\sigma_k(t)$ "
- Page 89, 3rd line : add: "Suppose the precedence constraints
can be represented by an inverted tree."
- Page 118, }
Page 119,) : for "6" read "*"
- Page 127, 5th line : for "to" read "t".



Abstract

This report reviews existing theory on the deterministic machine scheduling problem. The problem is formulated, the restrictions that are usually assumed in literature, are examined and several optimality criteria are compared and discussed. Known methods to attack the problem are described and exemplified. Certain situations receive special attention, in particular those where there are one, two or three machines, two jobs or a number of parallel identical machines. The report concludes with chapters on the general flow shop and job shop problem and on scheduling problems in economic reality. An extensive bibliography is included.

Contents

	<u>page</u>
1. Introduction	1
2. Formulation, definitions and criteria	4
2.1. Problem formulation	4
2.2. Restrictive assumptions	11
2.3. Optimality criteria	15
2.3.1. Criteria based on flow times and completion-dates	16
2.3.2. Criteria based on due-dates	19
2.3.3. Criteria based on inventory cost and utilization	21
2.3.4. Criteria based on change-over times	25
2.3.5. Multiple criteria	25
2.3.6. Conclusions	26
3. Methods of solution	28
3.1. Introduction	28
3.2. Complete enumeration (CE)	29
3.3. Integer and linear programming (IP)	30
3.4. Dynamic programming (DP)	38
3.5. Branch-and-bound methods (BB)	42
3.6. Combinatorial-analytical methods (CA)	44
3.7. Algebraic methods (A)	47
3.7.1. Schedule algebras	47
3.7.2. Relation algebras	53
3.8. Sampling techniques (ST)	60
3.9. Heuristic methods (H)	62
3.10. Conclusion	67
4. Some special cases	68
4.1. Introduction	68
4.2. The one-machine problem ($n 1$)	68
4.2.1. Criteria based on completion-dates and flow times	69

4.2.1.1. Precedence constraints	70
4.2.2. Criteria based on due-dates	73
4.2.2.1. Precedence constraints	80
4.2.2.2. Number of tardy jobs	81
4.2.3. Criteria based on machine utilization	82
4.2.4. Criteria based on change-over times	82
4.2.5. Multiple criteria	83
4.2.6. Multiple identical parallel machines	86
4.2.6.1. Precedence constraints	89
4.3. The two-machines problem ($n 2$)	89
4.3.1. The $n 2 F F_{\max}$ problem	90
4.3.2. The $n 2 F \bar{F}$ problem	93
4.3.3. The $n 2 G F_{\max}$ problem	93
4.3.4. Miscellaneous two-machine problems	94
4.4. The three-machines problem ($n 3$)	98
4.5. The two-jobs problem ($2 m$)	102
5. The general flow shop and job shop problem	106
5.1. Introduction	106
5.2. The $n m P F_{\max}$ problem	106
5.3. The $n m F F_{\max}$ problem	115
5.4. The $n m G F_{\max}$ problem	116
5.4.1. Elimination of infeasible sequences	117
5.4.2. Branch-and-bound methods	120
6. Scheduling in economic reality	134
6.1. Present situation	134
6.2. Future developments	140
Bibliography	142

Foreword

This report was written to serve as reference material during a week course on general problems of optimal sequencing, given at the Mathematical Centre in Amsterdam in August 1973.

I have tried to give a comprehensive survey of existing theory, that would be interesting both to relative laymen and more experienced mathematicians. The former category will perhaps want to skip some of the mathematical proofs; the latter category might be interested in a more mathematical version of this report that will appear in due course. Still, I feel the present mixed approach is fairly well suited to a problem that has such obvious real-life implications. I hope that any reader will at least understand why I think this seemingly easy problem so challenging and fascinating to study.

If the report accomplishes this and perhaps even functions as a basis for a continuing interest, I will be very happy. Naturally, I would welcome any criticism or additional remark that readers would want to make.

Thanks are finally due to David Brée for reading the manuscript, to Jan Karel Lenstra for stimulating conversations, to Elly van Buuren for the typing and to Happy for surviving it all.

Alexander Rinnooy Kan.

1. Introduction

This report aims to give a review of what has become known as the machine scheduling problem. This name covers a large class of various combinatorial and stochastic problems, all centered around the crucial question of the optimal sequence. We may as well state right at the beginning that we will deal exclusively with non-stochastic situations; this eliminates all theory on queues, waiting-lines etc. etc.. However, even within this smaller class, there is variation enough. This by itself leads to one of the major problems of scheduling research: there are so many sides to the problem, so many variations of it and so many ways to attack it, that the existing theory consists mostly of a great number of individual contributions lacking any interdependence or coherence. There simply is not a general theory where all these contributions could be fitted into. A first step in the right direction, however, might be made by gaining some insight in what has been done so far, in order to discover gaps, common traits and overlaps. This report is meant to be a modest contribution towards that goal.

Another aspect of the lack of a common language and theory is the confusing vocabulary and notation, found in scheduling literature. We shall give many definitions and notations in chapter 2. However, we point out straight away that we shall freely use the words "scheduling" and "sequencing" to designate the same activity whereby the processing order of a number of jobs by a number of machines is determined. Sometimes a difference is made between the two in that sequencing is supposed to give only the ordering itself, while scheduling explicitly gives starting times and completion times of all machine operations (i.e. Ashour [2], Elmaghraby [29]).

We assume, however, that once the processing order has been determined, the jobs will be finished in as short a time as possible, and therefore we do not need to distinguish between the two concepts.

Our interest in scheduling problems is mainly theoretical, which does not imply, of course, that we do not look for efficient ways to solve them - all combinatorial problems, being finite, are theoretically solvable by complete enumeration! This means one has to judge the quality of algorithms not (only) by looking at their mathematical beauty and elegance, but by looking at their computational performance. Although much obviously depends on the individual programmer and the computer used, we will try to give an impression of the results wherever this seems appropriate.

This report is organized along the following lines. In chapter 2 we formulate the problem, give notations and definitions of basic concepts and examine the many restrictions that are usually implicitly assumed in literature. Next, in chapter 3, we examine all known methods that have been used so far to solve the machine scheduling problem^{*)}. The reader of this chapter will notice that some methods (e.g. algebraic methods, integer programming methods) are dealt with in far greater detail than other ones (e.g. branch-and-bound methods, combinatorial methods). This is due to the fact that in the following chapters we do not refer any more to the former ones, while the latter ones are used so frequently that examples of their application can be found throughout the whole report.

In chapter 4 we deal with a few special cases where either the number of machines or the number of jobs is small, and an interesting theory has been developed. We do not avoid giving proofs, but do not give unduly lengthy or complex ones.

*) The only known method that we do not treat, is the general non-linear programming approach, advocated by Fisher (Lagrangian multipliers, [31]) and Nepomiastchy (penalty functions, [78]). It is too early to judge the usefulness of their approach.

Usually, they are not especially instructive and constitute mainly of checking if the proposed theorem holds true under all conceivable circumstances. The main purpose that could be served by publication of all these proofs, is to impress once more upon the reader the inadequacy of present combinatorial-analytical techniques for all but the simplest structured problems.

Chapter 5 then deals with the general problems; the best we can do here is to present a few elimination methods and a few numerical methods whereby an optimum might be found within a reasonable time.

Then, finally, in chapter 6 we take a look at the economic realisticness of the scheduling problem and suggest a few possible future developments.

We finish by giving a fairly large bibliography. Though it is not complete (as no bibliography ever is), we hope to have included all literature that is relevant at this moment.

2. Formulation, definitions and criteria

2.1. Problem formulation

The general formulation of the machine scheduling problem that we shall use here, is:

"Given n jobs that have to pass through m machines in a prescribed order under certain restrictive assumptions, what is then according to some criterium the optimal order in which each machine handles the jobs?"

We shall have to say more about the implications of this formulation in chapter 6. However, it should be clear that the problem was inspired by a typical real-life situation as it exists for instance in so-called job shops. There indeed each customer's order must be routed through the necessary machinery; materials, tools and labour must be allocated, processing and set-up times have to be estimated and a so-called due date is agreed upon by which the job(s) should be finished. Obviously the management of such an organisation is a complicated task, especially where so many different and related decisions have to be made continuously. The sequencing decision itself is preceded by planning activity and followed by control activity, both of them involving economic and technological judgments that strongly influence the sequencing decision itself.

The same complexity is characteristic of many other real-life situations where "machine scheduling problems" arise, albeit in a different context: the scheduling of classes to classrooms, classes to professors, hospital patients to test equipment, jobs to computers, cities to salesmen, dinners to cooks, homework to pupils etc. etc.. As to the effects of a good scheduling decision Mellor [66] quotes a list of no less than 27 goals that can be attained by good scheduling, with among them items as diverse as day-to-day stability of work force and anticipation of price changes!

Apart from this kind of complexity, many "local" circumstances, particular to a real-life situation, and perhaps cropping up while a number of jobs is already on its way, might cause a change in previously made decisions: a machine has broken down, a machine operator has become ill, an important client has placed an order which should get priority, a due date is being changed, etc. etc..

Obviously no theoretical analysis can take all these factors into account. The machine scheduling problem does not deal at all with questions of "what to produce?" and "how to produce?", but only with situations where decisions on these aspects have been previously made and will not be subject to change any more. Does economic reality justify this simplification? Is it ever really possible to separate the sequencing decision in this degree from other decisions? Pounds [80] reports that management is often not even aware that a sequencing problem exists; there are so many decisions to be made that the simple order in which each machine handles the jobs is not perceived as an influencable and relevant variable any more!

Still, the abstraction involved in the machine scheduling problem-formulation, can be defended in various ways; Elmaghraby [29] points out that sequencing decisions are likely to get more and more important as the computer takes over many routine decisions and as improved operations research techniques perfect other ones. Remembering also that it is only through study of components of a system, that we can gain understanding of the whole, it is not surprising to find that the abstract machine scheduling problem crops up in management science literature as early as the 1920's. The well-known concept of the Gantt chart, while no substitute for decision-making itself, at least presents available information about jobs and machines in a clear way and was one of the great innovations of the scientific management era.

The modern development of scheduling theory, however, where one tries to find an optimal sequence according to a well-defined criterium, has its starting point as late as 1954, when Johnson's classical paper on the two machine flow-shop [54] was published. Since then many different operations research techniques, most of which are mentioned in chapter 3, have been tried out on this problem with various degrees of success. Quite early the distinction between a deterministic and a stochastic approach to the scheduling problem was made; as was mentioned in chapter 1 we shall deal exclusively with the former situation.

In this second chapter we introduce the various notations to be used throughout this report. More specifically we pay attention (in 2.2.) to the rather heavy restrictions, that are usually assumed in existing literature, and to the various criteria whereby one can judge the qualities of a schedule (see 2.3.). First of all, however, we give basic definitions and notations, and a classification of scheduling situations. In all this we adapt ourselves mainly to the conventions of Conway, Maxwell and Muller [24] and of Said Ashour [2]. Let us first talk, then, of jobs, machines and operations.

A job (task, commodity, production lot, job lot) is obviously a product, produced by certain machines. There are n jobs to be considered^{*)}; they are designated by J_1, \dots, J_n or by job 1, job 2, ..., job n .

A machine (processor, resource, facility) is capable of performing one specific production process. There are m machines, designated M_1, \dots, M_m or machine 1, ..., machine m .

*) In general, we use capitals for solution-dependent variables and lower cast for initially given ones. The only exception is the use of capitals for J_1, \dots, J_n and for M_1, \dots, M_m .

A job J_k and a machine M_ℓ together uniquely determine an operation to be performed by M_ℓ on J_k and designated as (J_k, M_ℓ) or simply as (k, ℓ) . The set of all operations is the Cartesian product $\mathcal{Y} \times \mathcal{M}$ where $\mathcal{Y} = \{J_1, \dots, J_n\}$ and $\mathcal{M} = \{M_1, \dots, M_m\}$. Operations are the basic elements in the machine scheduling problem. With each operation (k, ℓ) is associated a real number $p_{k\ell}$, the processing time, indicating the amount of time it will take machine ℓ to complete work on job k , and including set-up time only in so far as these times are independent of the particular order in which machine ℓ handles the jobs. If $p_{k\ell} = 0$, this indicates that job k needs not to be attended to by machine ℓ .

Now an essential characteristic of the machine scheduling problem is that the order in which the jobs pass the machines is strictly prescribed by, say, technological considerations. That is to say: each subset

$$\{(J_k, M_1), \dots, (J_k, M_m)\} \quad (k = 1, \dots, n)$$

is strictly ordered by an ordering relation \ll :

$$(J_k, M_{i_1}) \ll (J_k, M_{i_2}) \ll \dots \ll (J_k, M_{i_m})$$

where (i_1, \dots, i_m) is some given permutation of $(1, \dots, m)$.

We say in the above case that (J_k, M_{i_1}) directly precedes (J_k, M_{i_2}) , etc., and we say that (J_k, M_{i_p}) precedes (J_k, M_{i_q}) whenever there is a chain of directly-precedes relations between them:

$$(J_k, M_{i_p}) < (J_k, M_{i_q}) \Leftrightarrow (J_k, M_{i_p}) \ll \dots \ll (J_k, M_{i_q}).$$

We can present the information about the route through the machines that each job k has to follow, in several ways. One possibility is combining all operations into an $m \times n$ matrix

called the job sequencing matrix S (Ashour [2]). For instance, suppose one has 3 jobs on 2 machines whereby job 1 has to pass through M_1 and M_2 (in that order), job 2 through M_2 and M_1 and job 3 through M_1 and M_2 , then S would look like this:

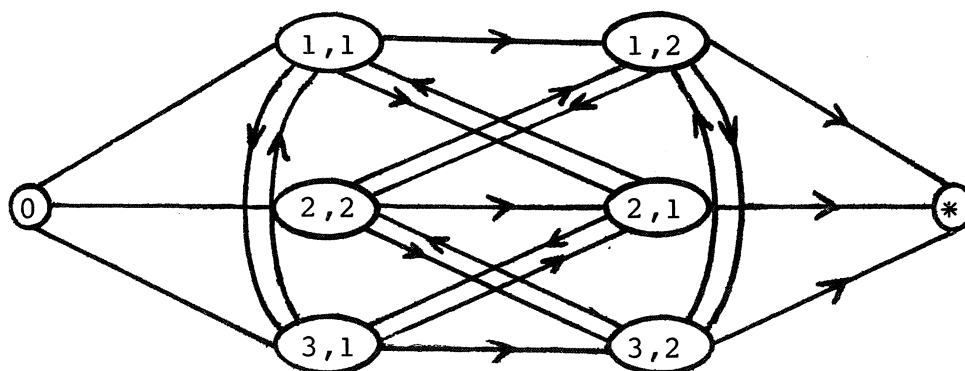
$$S = \begin{pmatrix} (J_1, M_1) & (J_1, M_2) \\ (J_2, M_2) & (J_2, M_1) \\ (J_3, M_1) & (J_3, M_2) \end{pmatrix}$$

or just simply:

$$S = \begin{pmatrix} (1,1) & (1,2) \\ (2,2) & (2,1) \\ (3,1) & (3,2) \end{pmatrix}$$

The rows of S convey all information on the routes of job 1, 2 and 3.

Another, very convenient way to present this information is in the form of a graph; usually two dummy operations are added to mark beginning and end of the whole process. Each node represents an operation, and a directed arc connects two nodes if the corresponding operations have a "directly-precedes" relationship, the direction of the arc corresponding to the direction of the job route. Furthermore, all operations performed on the same machine, i.e. the set $\{(J_1, M_\ell), \dots, (J_n, M_\ell)\}$, are usually connected by double directed arcs, whose significance will become apparent later on. These arcs are called disjunctive arcs and a graph of this type is usually called a disjunctive graph (Roy [86]). In our example the graph would look like this:



We will return to disjunctive graphs in 5.4..

Now there are several significantly different types of machine ordering per job. The simplest situation seems to exist when each job passes the machines in the same order (which we can, without loss of generality, assume to be $(1, \dots, m)$). In this situation we speak of a flow shop; we designate it by the letter F.

In a flow shop each job passes the machines in the same order, but that does not imply that each machine also handles the jobs in the same order. In fact, it is very likely that in some optimal sequence one job will "overtake" the other on some machine. If in a flow shop this "passing" is not permitted, we have a significantly easier problem; we designate this situation by the letter P.

If at least two jobs pass the machines in a different order (as in our previous example), we are in the most general situation. We then speak of a (general) job shop and use the letter G. In a job shop, each job has its particular route through the machines and these routes may all be different.

We have now laid the basis for a classification of machine scheduling problems, adapted from the one given by Conway, Maxwell and Miller [24]. The classification looks like this:

$$A|B|C_1, C_2, \dots |D|E$$

where:

- A = number of jobs (n in the general case);
- B = number of machines (m in the general case);
- C₁ = type of machine ordering per job (F, P or G);
- C₂ = any other relevant characteristics of the scheduling situation; for this, see the next paragraph (2.2.);
- D = the optimality criterium (for this, see 2.3.);
- E = the particular solution method employed (for this, see chapter 3).

E may be not present and is in fact mainly introduced here for use in the bibliography.

The example we have considered previously, would be classified as: 3|2|G|D, where D is the optimality criterium.

Our discussion so far permits a clearer formulation of the scheduling problem. The order of the jobs through the machines being given by technological requirements, the scheduling problem boils down to finding an ordering of the jobs on each machine, which is compatible with the technological requirements and which leads to an optimal schedule according to one of the criteria in 2.3..

The requirement of compatibility is indeed non-trivial. For suppose, in our previous example, we propose the solution that job 2 precedes job 1 on machine 1 and job 1 precedes job 2 on machine 2. We then have a contradiction:

$(J_2, M_2) \ll (J_2, M_1)$	(technological requirement)
$(J_2, M_1) \ll (J_1, M_1)$	(from above)
$(J_1, M_1) \ll (J_1, M_2)$	(technological requirement)
$(J_1, M_2) \ll (J_2, M_2)$	(from above)

implying

$$(J_2, M_2) \ll (J_2, M_1) \ll (J_1, M_1) \ll (J_1, M_2) \ll (J_2, M_2)$$

so that (J_2, M_2) would precede itself!

We conclude that we shall have to find efficient ways to eliminate these so-called infeasible sequences, and note in passing that above-mentioned incompatibility corresponds to a cycle in the disjunctive graph, where disjunctive arcs have been changed to normal directed ones in accordance with the proposed solution.

Before we take a look at the many ways in which a sequence might be optimal, we look at the severe underlying restrictions that have so far almost universally been assumed in scheduling literature.

2.2. Restrictive assumptions

In most of the existing literature on the machine scheduling problem, many restrictions are assumed to be valid. This, of course, increases the artificiality of the problem formulation into no insignificant degree. As we shall deal with criticism on these aspects of the formulation in a later chapter (i.e. chapter 6), we only repeat here the well-known defenses of the large degree of abstraction involved: namely, that this is unavoidable, and not essential, that it makes the problem more general and that it may well be relaxed in a more advanced state

of knowledge. Certainly it cannot be denied that even the highly stylized version of the scheduling problem is difficult enough to solve and that degree of applicability is not the only criterium by which to judge the value of mathematical analysis. Also the phenomenon of a developing branch of mathematics, being able to deal with more and more complicated situations, is well known from the past. However, the fact that so very few real life applications of scheduling theory are known, and the fact that, of the known applications, most employ heuristic (i.e. purposely suboptimal) methods ought to^{*)} worry mathematicians engaged in scheduling research, and merits the closer look that we shall take at this problem later on.

Many of the restrictions mentioned hereunder are automatically assumed in all existing literature; however, some articles distinguish themselves by dropping a few of them. The notation, introduced in 2.1., does permit an indication of this. Thereby we extend the notation of Conway, Maxwell and Miller [24]. We shall mention any restriction that is not assumed, designating it by its classification from the list below. A few examples of the extended notation will be given at the end of 2.2. and 3.1..

As to the list of all restrictive assumptions, there is an interesting duality between jobs and machines that we have tried to stress by the order of the items.

(J1) The set J of jobs is known and fixed.

(M1) The set M of machines is known and fixed.

(J2) All jobs are available at the same time (zero).

*) We realize that this is a subjective judgment.

- (M2) All machines are available at the same time (zero).
- (J3) All jobs remain available during an unlimited period (i.e. no due-dates).
- (M3) All machines remain available during an unlimited period (no labour-shortage, no break-down).
- (J4) Each job is in one of three states: waiting for the next machine, being operated by a machine or having passed its last machine.
- (M4) Each machine is in one of three states: waiting for the next job, operating on a job or having finished its last job.
- (J5) All jobs are different.
- (M5) All machines are different.
- (J6) All jobs are equally important.
- (M6) All machines are equally important^{*)}.

As to the interaction of jobs and machines, it is usually assumed that:

- (J7) Each job passes all the machines assigned to it.
- (M7) Each machine processes all the jobs assigned to it.
- (J8) Each job is processed by one machine at a time (i.e. no lap-phasing, no assembly).
- (M8) Each machine processes one job at a time.

*) I.e., no one can be missed or replaced by another one.

- (JM1) All processing times are known and fixed (i.e. sequence independent).
- (JM2) Each operation once started must be completed without interruption (no pre-emption, no job-splitting).

The asymmetry between jobs and machines is then due to:

- (JM3) The processing order of each job by all machines is known and fixed.
- (JM4) The processing order by each machine of all jobs is unknown and has to be fixed.

Many of these assumptions have been mentioned previously.

Obviously some of the assumptions have further reaching theoretical consequences than others. Simple assumptions like (J2) and (M2) can usually be dropped pretty easily. But assumptions like (J1) and (M1) are fundamental to a large part of scheduling theory: they distinguish the static (deterministic) problem approach from the dynamic (stochastic) one. As we shall deal exclusively with the former problem, these assumptions will not be dropped anywhere in this report. A good introduction to the entirely different theory of the dynamic case can be found in Conway, Maxwell and Miller [24], chapter 7 - 10.

It remains now to give a few examples of the extended notation.

- (i) A problem whereby n jobs are to be scheduled on one machine with sequence dependant set-up costs (assumption (JM1) is therefore not valid) will be designated as $n|1|(JM1)|D$ where D indicates some optimality criterium, e.g. minimum total set-up costs.

- (ii) A n -job, m -machine job shop problem, where job-splitting is allowed (see assumption (JM2)), will be designated as $n|m|G, (JM2)|D$, where D again is some optimality criterium.

We now turn to an investigation of optimality criteria.

2.3. Optimality criteria

When discussing optimality criteria, it is useful to classify them in a certain way. Although our theoretical interpretation of the scheduling problem is very restricted, so that we cannot introduce any criteria that suggest the interdependence of the scheduling decision and other ones regarding the production process, there still is a surprising variety of criteria to choose from. There are many ways to classify them.

We can distinguish between job-based criteria and machine- or shop-based criteria; we can distinguish between criteria based on completion-times and criteria based on due-dates (Gere [36]), or between criteria based on individual jobs and criteria based on the complete sequence (Elmaghraby [29]); we can also classify criteria according to whether they are time-based or cost-based, weighted or not-weighted (weights being attached to each job according to its importance, which implies dropping assumption (J6)) and single or multiple (Ashour [2]).

Now of these classifications is entirely satisfactory. However, for reasons of clarity, we have split the criteria up in five groups:

- (1) criteria based on completion-dates and flow-times;
- (2) criteria based on due-dates;
- (3) criteria based on inventory cost and the concept of utilization;

- (4) criteria based on change-over times;
- (5) multiple criteria.

We shall have more to say about the realisticness of these criteria in chapter 6. However, for the present this will suffice.

2.3.1. Criteria based on flow-times and completion-dates

We first define the relevant concepts. As usual we have n jobs J_1, \dots, J_n , m machines M_1, \dots, M_m , and $n \cdot m$ operations $\{(J_k, M_\ell)\}$ with processing times $p_{k\ell}$.

Now, let:

$r_k \stackrel{\text{def}}{=} \text{release date of } J_k \text{ (the earliest date that processing could start, which is equal to zero if assumption (J2) is not dropped);}$

$W_{k\ell} \stackrel{\text{def}}{=} \text{waiting time of } J_k \text{ before } M_\ell;$

$W_k \stackrel{\text{def}}{=} \sum_{\ell=1}^m W_{k\ell} \text{ (total waiting time for } J_k);$

$p_k \stackrel{\text{def}}{=} \sum_{\ell=1}^m p_{k\ell} \text{ (total processing time of } J_k);$

$C_k \stackrel{\text{def}}{=} \text{completion-date of } J_k \text{ (the date on which the last operation is finished);}$

$F_k \stackrel{\text{def}}{=} \text{flow-time of } J_k \text{ (the time } J_k \text{ spends in the shop).}$

There are a few elementary relations between these concepts:

$$C_k = r_k + W_k + p_k \quad (1)$$

$$F_k = W_k + p_k \quad (2)$$

$$C_k = r_k + F_k \quad (3)$$

We can now define a number of frequently used criteria, based on these definitions:

- (1) minimize the maximum completion-date $C_{\max} = \max_k \{C_k\}$;
- (2) minimize the maximum flow-time $F_{\max} = \max_k \{F_k\}$
(this criterium is by far the most frequently used one);
- (3) minimize the maximum waiting-time $W_{\max} = \max_k \{W_k\}$;
- (4) minimize the average completion-date $\bar{C} = \frac{1}{n} \sum C_k$;
- (5) minimize the average flow-time $\bar{F} = \frac{1}{n} \sum F_k$;
- (6) minimize the average waiting-time $\bar{W} = \frac{1}{n} \sum W_k$.

Now, (4), (5) and (6) are really special cases of:

- (7) minimize the weighted sum of completion-dates $\sum \alpha_k C_k$, where α_k indicates the relative importance of J_k (dropping assumption (J6));
- (8) minimize the weighted sum of flow-times $\sum \alpha_k F_k$;
- (9) minimize the weighted sum of waiting-times $\sum \alpha_k W_k$.

However, we have from (2) and (3):

$$\sum \alpha_k F_k = \sum \alpha_k W_k + \sum \alpha_k p_k$$

$$\sum \alpha_k C_k = \sum \alpha_k r_k + \sum \alpha_k F_k$$

so, $\sum \alpha_k p_k$ and $\sum \alpha_k r_k$ being sequence-independent constants, (7), (8) and (9) are equivalent criteria, as are (4), (5) and (6).

However, the C_{\max} and F_{\max} criterium need not be identical, unless of course $r_k = 0$ for all k , in which case $C_k = F_k$.

Also, the F_{\max} -criterium does not need to be equivalent to the \bar{F} -criterium.

Example: suppose we have a $2|3|G|F_{\max}$ problem with matrix S :

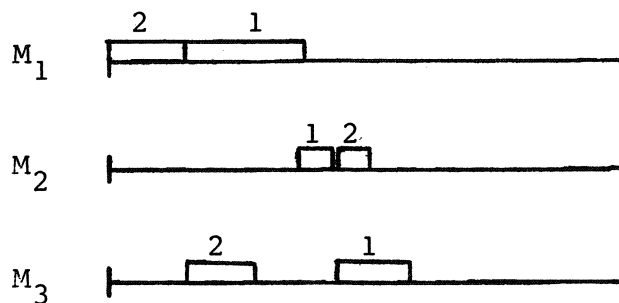
$$S = \begin{pmatrix} (1,1) & (1,2) & (1,3) \\ (2,1) & (2,3) & (2,2) \end{pmatrix}$$

and processing times $p_{k\ell}$:

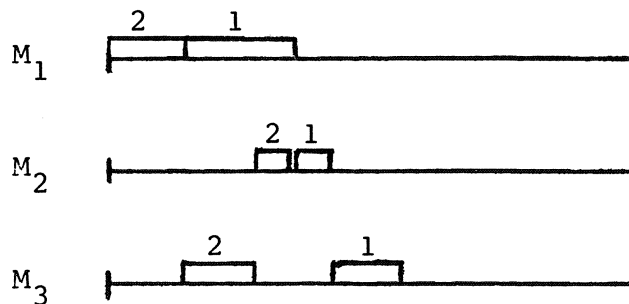
$$p_{11} = 3 \quad p_{12} = 1 \quad p_{13} = 2$$

$$p_{21} = 2 \quad p_{22} = 1 \quad p_{23} = 2$$

Using the well-known concept of a Gantt-chart to depict possible sequences, we find two optimal sequences where $F_{\max} = 8$:



and



However, in the first sequence $\bar{F} = \frac{1}{2}(8+7) = 7\frac{1}{2}$ and for the second one $\bar{F} = \frac{1}{2}(8+5) = 6\frac{1}{2}$.

Equivalent properties of many optimality criteria led to the concept of a regular measure (Conway, Maxwell and Miller [24]). This is a function of the completion-dates $\phi(C_1, \dots, C_n)$ that is monotone in each variable:

$\phi(C_1, \dots, C_n) \geq \phi(C'_1, \dots, C'_n) \Leftrightarrow C_k \geq C'_k$ for at least one k .

$C_{\max}, F_{\max}, W_{\max}, \bar{C}, \bar{F}, \bar{W}, \sum \alpha_k C_k, \sum \alpha_k F_k$ and $\sum \alpha_k W_k$ are all regular measures.

Usually we shall assume that $r_k = 0$ for all k , in accordance with assumption (J2), and that therefore $F_k = C_k$. Any departure from this convention will be clear from the context.

2.3.2. Criteria based on due-dates

We drop assumption (J3) and assume due-dates d_k have been set for each job J_k . We can now define:

$$L_k \stackrel{\text{def}}{=} C_k - d_k \quad (\text{the lateness of } J_k)$$

$$T_k \stackrel{\text{def}}{=} \max(0, L_k) \quad (\text{the tardiness of } J_k)$$

$$E_k \stackrel{\text{def}}{=} \max(0, -L_k) \quad (\text{the earliness of } J_k).$$

Here we have the elementary relation.

$$T_k - E_k = L_k.$$

Again, we can define a number of optimality criteria:

$$(10) \quad \text{minimize } L_{\max} = \max_k \{L_k\};$$

$$(11) \quad \text{minimize } T_{\max} = \max_k \{T_k\};$$

$$(12) \quad \text{maximize } E_{\max} = \max_k \{E_k\};$$

$$(13) \quad \text{minimize } \bar{L} = \frac{1}{n} \sum L_k;$$

$$(14) \quad \text{minimize } \bar{T} = \frac{1}{n} \sum T_k;$$

$$(15) \quad \text{maximize } \bar{E} = \frac{1}{n} \sum E_k;$$

$$(16) \quad \text{minimize } \sum \alpha_k L_k;$$

$$(17) \quad \text{minimize } \sum \alpha_k T_k;$$

$$(18) \quad \text{maximize } \sum \alpha_k E_k;$$

Now (13), (14) and (15) are again special cases of (16), (17) and (18) respectively.

Furthermore, we find by definition:

$$\sum \alpha_k L_k = \sum \alpha_k C_k - \sum \alpha_k d_k ,$$

so that, $\sum \alpha_k d_k$ being a sequence-independent constant, (13) is equivalent to (4), (5) and (6) and (16) is equivalent to (7), (8) and (9).

No such easy formulas exist for tardiness and earliness. Still, especially the former is a very realistic criterium; often the only concern of management is to finish a job on time or failing that, as soon as possible after the due-date. There is no extra premium in that situation on being finished well ahead of the due-date.

2.3.3. Criteria based on inventory cost and utilization

We may judge the quality of a schedule by looking more closely at what happens in the shop during the whole production process. Important measures to be considered are then:

$N_f(t)$ def number of jobs finished at time t ;

$N_w(t)$ def number of jobs waiting to be processed at time t ;

$N_p(t)$ def number of jobs actually being processed at time t ;

$A_f(t)$ def work finished, i.e. sum of the processing times of all operations finished at time t ;

$A_w(t)$ def work remaining, i.e. sum of the processing times of all operations that still have to be performed at time t ;

$A_p(t)$ def work in progress, i.e. sum of the processing times of all operations performed at time t .

By definition:

$$N_f(t) + N_w(t) + N_p(t) = n$$

$$A_f(t) + A_w(t) + A_p(t) = \sum_{k,l} p_{kl}$$

Now, if we consider all averages to be taken over the period $(0, F_{\max})$, we see that:

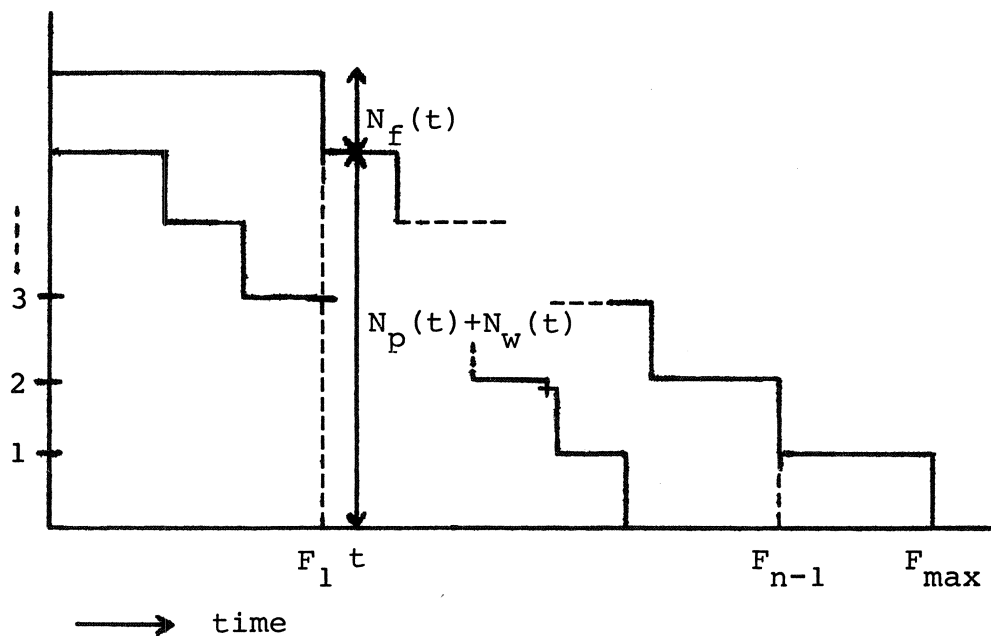
(1) $\bar{N}_p + \bar{N}_w$ gives an indication of average in-process inventory costs:

(2) \bar{N}_f gives an indication of average inventory costs for finished products;

(3) \bar{A}_p should be high, and depends heavily on the average length of $p_{k\ell}$.

By looking at the illustration below, where the jobs are started and finished in order (1, ..., n), we see directly that in this case the following relation holds:

$$\bar{N}_p + \bar{N}_w = (F_1 + \dots + F_n)/F_{\max} = n \bar{F}/F_{\max} \quad (4)$$



It is not difficult to prove that the same type of relation holds if $r_k \neq 0$ for all k and if the jobs are not completed in arrival order (see Conway, Maxwell and Miller [24], page 15-20). (In fact, all these relations are special cases of the fundamental equation of dynamic scheduling theory:

$$\bar{N}_p + \bar{N}_w = \lambda \bar{F}$$

where λ is the mean rate of job arrival. This equation holds true under very general circumstances).

We return again to (1). It is trivial to prove in the static case:

$$\bar{N}_p = \frac{\sum_{k,e} p_{k\ell}}{F_{\max}} \quad (3)$$

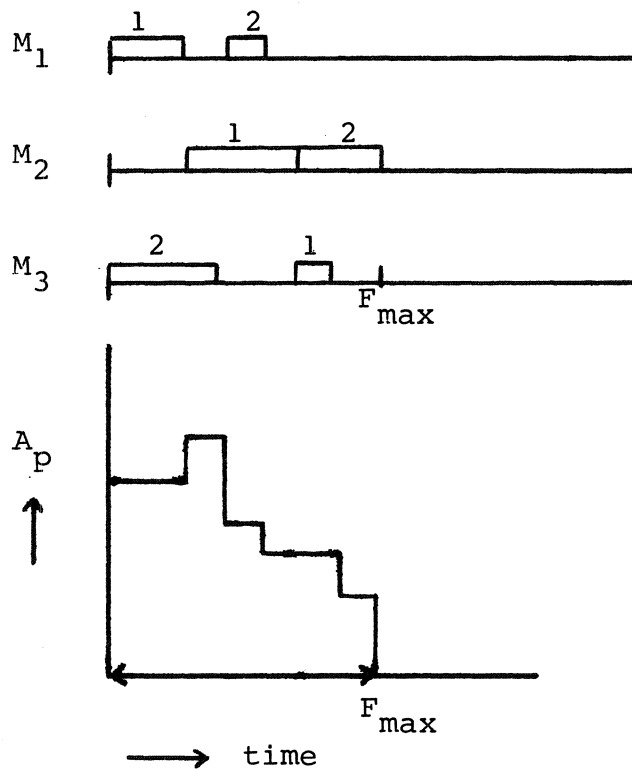
so we conclude:

$$\bar{N}_w = \frac{n \bar{F} - \sum_{k,e} p_{k\ell}}{F_{\max}} = \frac{n \bar{W}}{F_{\max}} \quad (4)$$

and

$$\bar{N}_f = n - (\bar{N}_p + \bar{N}_r) = \frac{n(\bar{F} - F_{\max})}{F_{\max}} = n \frac{\bar{F}}{F_{\max}} - n.$$

As to \bar{A}_f , \bar{A}_r and \bar{A}_p , it is not so easy to derive comparably simple formulas for \bar{A}_f and \bar{A}_w *). We can easily, however, construct the graph of $A_p(t)$ for an example Gantt chart below.



*) In fact, \bar{A}_f can be written as a complicated weighted sum of the $w_{k\ell}$, but this does not seem to lead anywhere.

It is easy to verify from this drawing that

$$\bar{A}_p = \frac{\sum_{k,l} p_{kl}^2}{F_{\max}} \quad (5)$$

so that any sequence minimizing F_{\max} maximizes \bar{A}_p .

There are two further measures here that require attention. However, if we define idle time I_ℓ on machine ℓ to be the time that the machine is not used between 0 and F_{\max}^* , it appears that the sum of idle times is equal to

$$m F_{\max} - \sum_{k,l} p_{kl}$$

so that minimization of F_{\max} ensures minimal (weighted) idle times.

A more important measure is that of utilization, which reflects the necessity of intensive use of available machinery because of fixed costs caused by depreciation allowance etc..

Utilization is usually defined as

$$\bar{U} = \frac{\sum_{k,l} p_{kl}}{m F_{\max}}$$

which implies again that maximum mean utilization is equivalent to minimum F_{\max} .

Combining this with (4), we get:

$$\bar{U} = \frac{(\sum_{k,l} p_{kl}) \cdot (\bar{N}_p + \bar{N}_w)}{m n \bar{F}} = \frac{\bar{p}(\bar{N}_p + \bar{N}_w)}{\bar{F}}$$

where p is average processing time; this equation again plays a fundamental role in dynamic theory.

*) This definition is not used by Ashour [2], which leads to an error on page 51.

2.3.4. Criteria based on change-over times

For the sake of completeness we have added criteria based on change-over costs to own list. This criterium implies assumption (JM1) is partly dropped. In fact, only one criterium has been extensively studied, namely the minimization of total change-over times in the $n|1$ situation, where these times c_{ij} - when changing from job i to job j -, are sequence-dependent. This problem is equivalent to the well-known travelling-salesman problem; we shall return to the subject in 4.2.4.. One could view a $n|m|F$ problem, where the object is to minimize total sequence-dependent change-over costs, as an extended travelling-salesman problem, where each "city" gets visited in the same sequence by all the (more and more experienced) salesmen!

The other situation of interest is the situation wherein we have to satisfy a given continuous demand for several products, produced by one machine. The object then becomes to minimize the number of change-overs in a certain time-interval (Glassey [41]). We will return to this problem in 4.2.4..

2.3.5. Multiple criteria

In actual situations it happens frequently that we have to take into account not one, but several criteria at the time. This leads to general problems of decision-making with multiple objectives. We have to combine all the objectives into one measure whereby one can judge alternative outcomes. Several general methods have been developed so far (see the review by Roy [87]). One could, for instance, order all possible outcomes lexicographically, i.e. completely order the objectives, choose the outcome which scores highest on the first objective, break ties by means of the second objective etc.; or one could attach weights α_k to each objective O_k and combine them into a linear function $\sum \alpha_k O_k$; alternatively one could get goals for each

objective and try to minimize the (weighted) sum of the differences between goal and actual value of each objective function, etc. etc.*). There is no doubt that multiple objective decision-making is a frequently occurring problem, especially in strongly areas like scheduling where decisions are influenced by many factors. However, there are doubts about the applicability of the afore-mentioned mathematical methods, and in any case little scheduling research has been conducted along these lines. In fact, only two studies are known, one by Smith [93] and one by Florian et al. [17], where \bar{F} respectively F_{\max} is minimized under the side condition that $T_{\max} = 0$. We will return to these studies in 4.2. too.

2.3.6. Conclusions

We conclude this section by giving a short review of all criteria mentioned so far. We have split them up in equivalent groups, equivalence meaning that the same sequence(s) is (are) optimal for all criteria in the group.

The groups are:

- (1) C_{\max}
- (2) $F_{\max}, \bar{N}_p, \bar{A}_p, \bar{U}, \Sigma I_\ell$
- (3) W_{\max}
- (4) $\bar{C}, \bar{F}, \bar{W}, \bar{L}$
- (5) $\Sigma \alpha_k C_k, \Sigma \alpha_k F_k, \Sigma \alpha_k W_k, \Sigma \alpha_k L_k$

*) Ashour [2] gives a worked-out example of several techniques.

- (6) L_{\max}
- (7) T_{\max}^*
- (8) E_{\max}
- (9) \bar{T}
- (10) $\sum \alpha_k T_k$
- (11) \bar{E}
- (12) $\sum \alpha_k E_k$
- (13) $\bar{N}_p + \bar{N}_w, \bar{N}_f, \bar{F}/F_{\max}$
- (14) $\bar{N}_w, \bar{W}/F_{\max}$

If we look at this list, it is not so surprising that most work has been done so far on groups (2), (4) and (5).

A comparison of all these criteria would be interesting; the only studies we know of are by Gupta [45], and by Ashour [2] (for just one example). We will return to the former study in chapter 6.

*) Actually, the sequence minimizing L_{\max} also minimizes T_{\max} (but not necessarily vice versa): if

$$L_{\max}(s') \leq L_{\max}(s)$$

for all sequences s , then:

$$\max(0, L_{\max}(s')) = T_{\max}(s') \leq \max(0, L_{\max}(s)) = T_{\max}(s)$$

for all s .

3. Methods of solution

3.1. Introduction

The machine scheduling problem is a typically combinatorial optimization problem where the optimum is to be found among a large, but finite number of possible solutions.

Most methods to attack this kind of problem typically try to reduce the set FS of all feasible schedules to a smaller set POS of potentially optimal schedules and look for the optimum within this smaller set. No general efficient method has so far been developed, the discreteness and the resulting "discontinuity" of the optimality criterium function leading to very difficult problems.

The machine scheduling problem belongs to a group of problems that center around the concept of an "optimal sequence". In his book [70], devoted to these problems, Müller-Merbach mentions four general solution methods for these problems:

- (1) complete (explicit) enumeration;
- (2) tree searching algorithms;
- (3) heuristic methods;
- (4) special algorithms.

Now (2), according to Müller-Merbach, **consists** of the following methods:

- (2a) dynamic programming;
- (2b) branch-and-bound procedures;
- (2c) implicit (bounded) enumeration,

and (3) can also be further split up:

- (3a) non-iterative methods;
- (3b) iterative methods,

(3a) and (3b) usually being used simultaneously. (4) consists of a few special algorithms that have been developed with analytical methods.

We shall not pay specific attention to (2c), by which Müller-Merbach means any technique by which a (heuristically found) solution is being gradually improved. Only a few applications of this method are known in machine scheduling; anyhow, the methodological distinction between (2c) and (2b) is not at all clear. We shall in what follows pay attention then to (1), (2a), (2b), (3) and (4), where (4) shall be interpreted as to include all combinatorial-analytical theory available on the machine scheduling problem. Furthermore, we shall remark on the application of integer and linear programming techniques to the machine scheduling problem and we shall study two methods that have been specifically developed within the machine scheduling context, namely the algebraic methods of Giffler [37] and Rial [83] and the application of sampling techniques by Heller [49] and other researchers.

Mainly for use in the bibliography abbreviations for each solution method are proposed in the heading of the section describing it; i.e. CE = complete enumeration, etc..

$n|m|G|F_{\max}|CE$ would then indicate a complete enumeration solution to the $n|m|G|F_{\max}$ problem.

3.2. Complete enumeration (CE)

We can be short on the subject of complete enumeration. In the $n|m|G$ problem there are $(n!)^m$ possible schedules, a number that soon reaches astronomic proportions. For instance, $(5!)^5 \approx 3 \cdot 10^{10}$, which implies that if a computer would evaluate 100.000 schedules a second, it would still take $3 \cdot 10^5$ seconds or approximately 1 year of computing time to evaluate all of them.

In the $n|m|F$ problem, there are for theoretical reasons in some cases (depending on the criterium) "only" $(n!)^{m-2}$ schedules to enumerate ($m > 3$), whereas in the $n|m|P$ problem, the order of the jobs on each machine is identical and therefore only $n!$ different schedules have to be evaluated. However, this number also soon outgrows any computer-feasible size.

It may be noted, however, that of the $(n!)^m$ different schedules of the general job shop problem, many will be infeasible because of incompatible job- and machine-orderings. Supposing we have an efficient algorithm to eliminate these infeasible schedules, could we then enumerate the remaining ones? In general the answer is no, since the number of feasible schedules n_F is bounded by can be quite high as well. In an $n|m|G$ problem, where each job passes all machines, the situation closest resembling the $n|m|F$ problem (where all $(n!)^m$ sequences are feasible) is the one in which all jobs pass all machines in the same order $(1, \dots, n)$ except for one job which passes machine $(\ell+1)$ before machine ℓ . This leads to $(n-1)$ unfeasible schedules: a very small reduction indeed!

3.3. Integer and linear programming (IP)

There have been several attempts to solve the machine scheduling problem by formulating it as an integer programming problem, which in the most general form looks like this:

minimize $c_1X + c_2Y$
subject to:

$$A_1X + A_2Y \leq b$$

$$X \geq 0$$

$$Y \geq 0, \text{ integer.}$$

For a general survey of integer programming, see Beale [12], Balinski [11] or Geoffrion [35].

The oldest attempts to solve the machine scheduling problem along these lines are by Bowman [16] and Wagner [102].

Bowman uses 0 - 1 variables X_{ijk} where $X_{ijk} = 1$ indicates that job i is processed on machine j in period k .

This leads to restraints of the type:

$$\sum_{k=1}^{\tau} X_{ijk} = p_{ij} \quad (i = 1, \dots, n; j = 1, \dots, m) \quad (1)$$

where τ is the scheduling period. If job splitting is not allowed, constraints of the type:

$$p_{ij} (X_{ijk} - X_{ij,k+1}) + \sum_{\ell=k+2}^{\tau} X_{ij\ell} \leq p_{ij} \\ (i = 1, \dots, n; j = 1, \dots, m; \\ k = 1, \dots, \tau) \quad (2)$$

have to be added, so as to prevent a 1-variable to be followed by a 0-variable and a 1-variable in that order.

As each machine may only handle one job at the time, we have constraints:

$$\sum_{i=1}^n X_{ijk} \leq 1 \quad (j = 1, \dots, m; k = 1, \dots, \tau) \quad (3)$$

The prescribed machine ordering for each job is reflected by constraints of the type:

$$p_{ij_1} X_{ij_2}^k \leq \sum_{\ell=1}^{k-1} X_{ij_1}^\ell \quad (k = 1, \dots, \tau) \quad (4)$$

for every given direct-precedence relation $(i, j_1) \ll (i, j_2)$.

Bowman suggests an optimality criterium function of the form:

$$1. \sum_{i=1}^n X_{ij_i}^t + 4. \sum_{i=1}^n X_{ij_i}^{t+1} + \dots + 4^{\tau-t} \sum_{i=1}^n X_{ij_i}^\tau$$

where j_i is the last machine job i has to pass through and

$$t = \max_i \sum_{j=1}^m p_{ij}.$$

The number of variables equals $n\tau$, and the number of constraints equals $m[n(2\tau+1) + \tau]$ for the general problem. Apart from the curious optimality criterium, it is clear that the number of 0 - 1 variables is excessively large and that this formulation could hardly be called practical.

The reason to mention this approach here is that a similar formulation by Von Lanzener and Himes [48] forms the only possibly successful linear programming approach to the problem.

We again have constraints (1) and (3), but job splitting is prevented here by introducing variables Y_{ijk} , where

$$Y_{ijk} = \begin{cases} 1 & \text{if } X_{ijk} - X_{ij,k+1} = 1 \\ 0 & \text{otherwise} \end{cases}$$

and demanding:

$$Y_{ijk} \geq X_{ijk} - X_{ij,k+1}$$

$$\sum_{k=1}^T Y_{ijk} = 1$$

Direct-precedence relations (i, j_1) (i, j_2) are reflected by

$$\sum_{k=1}^t Y_{ij_1 k} - \sum_{k=1}^{t+1} Y_{ij_2 k} > 0 \quad (t = 1, \dots, \tau)$$

Now we need to make sure that job i is processed on one machine at a time:

$$\sum_{j=1}^m X_{ijk} \leq 1 \quad (i = 1, \dots, n; k = 1, \dots, \tau)$$

We can reduce all summations by restricting them to feasible time periods. We can now use Bowman's criterium again. The essential point to notice now is that, by introducing Y_{ijk} , we have succeeded in making all coefficients equal to +1 or -1. We would therefore not be very surprised if a linear programming algorithm applied to this problem, would produce an integer solution, just as happens in the case of a transportation problem. However, computing experience with this algorithm is small, and an integer solution cannot be guaranteed. The latter fact reduces this algorithm effectively to a heuristic (suboptimal) one.

Wagner's approach is quite different and in fact only suitable for the $n|m|P$ problem. We give a formulation for the $n|3|P$ problem, where "only" $n!$ sequences have to be considered.

The permutation is determined by 0 - 1 variables X_{ij} where $X_{ij} = 1$ indicates job i comes in position j , with

$$\sum_{i=1}^n X_{ij} = 1 \quad (j = 1, \dots, n) \quad (5)$$

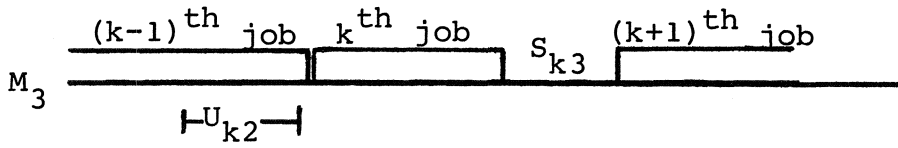
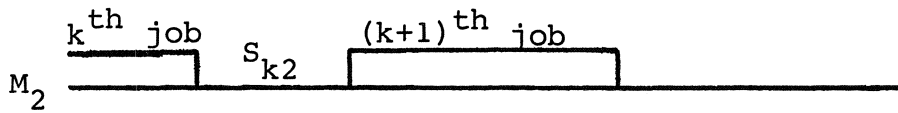
$$\sum_{j=1}^n X_{ij} = 1 \quad (i = 1, \dots, n) \quad (6)$$

To ensure that jobs are processed by one machine at the time in the right order, and that one machine only processes one job at the time, variables $S_{i\ell}$ and $U_{i\ell}$ are introduced where

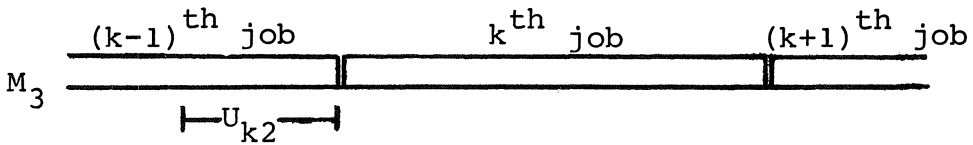
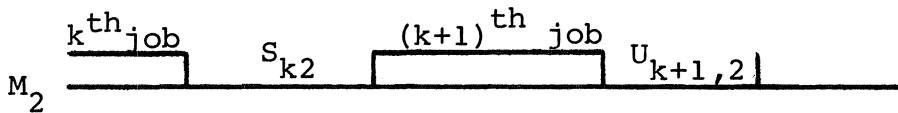
$$S_{i\ell} = \text{idle time on machine } \ell \text{ between the } i^{\text{th}} \text{ job and the } (i+1)^{\text{th}} \text{ job;}$$

$U_{i\ell}$ = waiting time of the i^{th} job between machine ℓ and machine $(\ell+1)$

and constraints of the following type (see the drawing):



or



$$S_{k2} + \sum_{i=1}^n p_{i2} X_{i,k+1} + U_{k+1,2} = U_{k2} + \sum_{i=1}^n p_{i3} X_{ik} + S_{k3} \quad (k = 1, \dots, n-1) \quad (7)$$

$$\sum_{i=1}^n p_{i1} X_{i,k+1} + U_{k+1,2} = U_{k,1} + \sum_{i=1}^n p_{i2} X_{ik} + S_{k2} \quad (k = 1, \dots, n-1) \quad (8)$$

Wagner chooses to minimize

$$\sum_{i=1}^n (p_{i1} + p_{i2}) X_{i1} + \sum_{i=1}^{n-1} S_{i3}$$

which is equal to $F_{\max} - \sum_{i=1}^n p_i$. The number of constraints is $(4n - 3)$. He tries to solve the problem by using an all-integer dual algorithm, created by Gomory. In chapter 14 of the book by Muth and Thompson [96], however, he has to report that he has "not yet found an integer programming method that can be relied upon to solve most machine sequencing problems rapidly".

A much better formulation is given by Manne [64]. He solves the $n|1||F_{\max}$ problem by using variables T_k to indicate the starting time of job k . (Manne restricts himself to integer T_k , but they may as well be real). Writing p_{11}, \dots, p_{n1} as p_1, \dots, p_n , the fact that job j either takes place before or after job k , is indicated by:

$$T_j - T_k \geq p_k \quad \text{or} \quad T_k - T_j \geq p_j \quad (9)$$

This is converted into one inequality by using 0 - 1 variables Y_{jk} and a constant C which should be larger than all possible values of T_j ($j = 1, \dots, n$).

Now the restrictions:

$$(C + p_k) Y_{jk} + (T_j - T_k) \geq p_k \quad (10)$$

$$(C + p_j)(1 - Y_{jk}) + (T_k - T_j) \geq p_j \quad (11)$$

are together equivalent to (9): if $Y_{jk} = 0$, (10) becomes $T_j - T_k \geq p_k$, and (11) is trivially true; if $Y_{jk} = 1$ we get $T_k - T_j \geq p_j$.

Other precedence relations (assumption (J6) being dropped), such as job j precedes job k , are given by trivial inequalities:

$$T_k - T_j \geq p_j \quad (12)$$

etc. etc.. Due-dates can also be incorporated:

$$T_k \leq d_k - p_k \quad (13)$$

Putting

$$T - T_k \geq p_k \quad (k = 1, \dots, n)$$

we can then minimize T .

Manne gives no computing results, and only indicates vaguely that this approach could be generalized to the $n|m|G|F_{\max}$ problem. However, this is trivial: taking $T_{k\ell}$ as the starting time of operation (J_k, M_ℓ) we have

$$T_{j\ell} - T_{k\ell} \geq p_{k\ell} \quad \text{or} \quad T_{k\ell} - T_{j\ell} \geq p_{j\ell} \quad (14)$$

for all pairs j, k^*). The prescribed machine order for each job is given by

$$T_{k\ell} - T_{km} \geq p_{km} \quad (15)$$

for every directly-precedes relation $(k, m) \ll (k, \ell)$.

Again we can introduce due-dates:

$$T_{kj_k} \leq d_k - p_{kj_k} \quad (k = 1, \dots, n) \quad (16)$$

where j_k is the last machine for job k (dropping assumption (J3)) and we can also easily drop assumptions (J2), (M2) and (M3):

$$T_{ki_k} \geq r_k \quad (17)$$

*) This can easily be generalized to the situation where a job does not necessarily pass through all the machines.

$$T_{k\ell} \geq e_{\ell} \quad (k = 1, \dots, n) \quad (18)$$

$$T_{k\ell} \leq f_{\ell} \quad (k = 1, \dots, n) \quad (19)$$

where (18) and (19) indicate the limited availability of machine ℓ and i_k is the first machine of job k . Sequence dependent set-up times c_{jkl} (when job k follows job j on machine ℓ) can be easily introduced in (14), so that assumption (JM1) can also be dropped.

The inequalities under (14) can again be combined into one inequality by introducing the 0 - 1 variable Y_{jkl} and a large constant C and demanding

$$(C + p_{k\ell}) Y_{jkl} + (T_{j\ell} - T_{k\ell}) \geq p_{k\ell} \quad (20)$$

$$(C + p_{j\ell})(1 - Y_{jkl}) + (T_{k\ell} - T_{j\ell}) \geq p_{j\ell} \quad (21)$$

We can then minimize T where

$$T - T_{kj\ell} \geq p_{kj\ell} \quad (k = 1, \dots, n)$$

and have a mixed-integer programming problem.

This formulation is given by Balas [7], Gupta [47] and Raimond [81]. Balas solve the problem by his more generally applicable filtermethod and Raimond uses a direct-search method; however, both methods effectively boil down to a branch-and-bound method, which in the case of Balas is introduced in another article by him (Balas [8]).

We think one may safely conclude by now that the elegant formulation of scheduling (and so many other) problems by means of 0 - 1 variables insufficiently takes into account the special structure of the scheduling situation. Therefore it is highly

unlikely that a general integer programming method will ever provide the most efficient way to solve scheduling problems.

3.4. Dynamic programming (DP)

There is no need to describe in detail here the familiar method of dynamic programming, due to Bellman. Good examples can be found in Beckmann [13] and Müller-Merbach [70]. Applications of this technique to general sequencing problems are quite numerous, but to the machine scheduling problem they are comparatively rare. We shall give an example, due to Lawler and Moore [59], which demonstrates the usefulness of the approach for a series of $n|1$ problems. Suppose jobs J_1, \dots, J_n , to be performed in this order, can be handled in two different ways. In the first way J_k requires g_k units of time, and a loss of $\gamma_k(t)$ is incurred upon completion of J_k at time t ; in the second way the time required is s_k units and the loss $\sigma_k(t)$. We want to minimize the total loss.

Now let

$f(k,t)$ = minimum total loss for first k jobs, job k
being finished no later than t .

By a typical dynamic programming argument, we see:

$$f(k,t) = \min \begin{cases} f(k,t-1) \\ \gamma_k(t) + f(k-1,t-g_k) \\ \sigma_k(t) + f(k-1,t-s_k) \end{cases} \quad (k = 1, \dots, n; t \geq 0) \quad (19)$$

We put:

$$f(0,t) = 0 \quad (t \geq 0)$$

$$f(k,t) = \infty \quad (k = 0, \dots, n; t < 0)$$

and solve our problem by calculating $f(n,T)$ where T is sufficiently large (e.g., $T = \sum_k \max(g_k, s_k)$).

A small example will clarify this method. Suppose we have two jobs J_1, J_2 ; $g_1 = 2, g_2 = 1, s_1 = 1, s_2 = 2$; $\gamma_k(t) = 2t, \sigma_k(t) = 3t$. Taking $T = 2 + 2 = 4$, we find:

$$\begin{array}{l}
 f(2,1) = \infty \\
 f(2,2) = \min \begin{cases} 4 + f(1,1) = \textcircled{7} \\ 6 + f(1,0) = \infty \end{cases} \\
 f(2,3) = \min \begin{cases} 6 + f(1,2) = 9 \\ 9 + f(1,1) = 12 \end{cases} \\
 f(2,4) = \min \begin{cases} 8 + f(1,2) = \textcircled{11} \\ 8 + f(1,3) = \min \begin{cases} 14 + f(1,1) = 17 \\ 17 + f(1,2) = 20 \end{cases} \\ 12 + f(1,2) = \min \begin{cases} 12 + f(1,1) = \min \begin{cases} 12 + f(1,0) = \infty \\ 14 + f(0,-1) = \infty \\ 15 + f(0,0) = \textcircled{15} \end{cases} \\ 16 + f(0,0) = 16 \\ 18 + f(0,1) = 18 \end{cases} \end{cases}
 \end{array}$$

from which we see: $f(2,4) = \min(7, 11, 15) = 7$, reached by producing J_1 in the second way and J_2 in the first way; J_1 is ready at $t = 1$, J_2 is ready at $t = 2$ and the costs are $1 \times 1 + 3 \times 2 = 7$.

We apply this to a $n|1$ problem. Suppose n jobs have processing times p_k and a common deadline d , and suppose we have loss functions

$$c_k(t) = \begin{cases} \alpha_k t & (t \leq d) \\ \beta_k & (t > d) \end{cases}$$

We want to determine a sequence so that $\sum_{k=1}^n c_k(t)$ is minimized.

This boils down to partitioning the jobs in two classes: those that will be completed on or before d and those that will be tardy. The first group will be sequenced according to the ratio's p_k/α_k (the job with the smallest ratio first - see 4.2.1.), the second group follows in arbitrary order.

We can solve this problem by ordering the jobs by their p_k/α_k ratio, putting

$$\begin{aligned} g_k &= p_k & \gamma_k(t) &= \alpha_k t \\ s_k &= 0 & \sigma_k(t) &= \beta_k \end{aligned}$$

and applying (19).

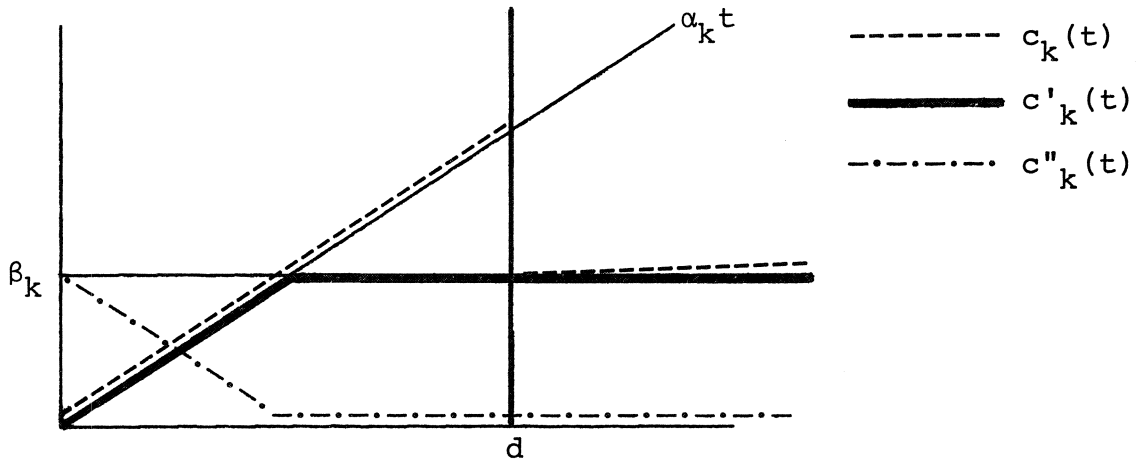
Now suppose α_k is given and a deadline d_k is given for every job. We put $\beta_k = \alpha_k d_k$ and choose d so that $\alpha_k d > \beta_k$ ($k = 1, \dots, n$). Then the sequence minimizing $\sum_k c_k(t)$ also minimizes $\sum c'_k(t)$ where

$$c'_k(t) = \min \{ \alpha_k t, \beta_k \}$$

so it maximizes $\sum c''_k(t)$ where

$$c''_k(t) = \max \{ \alpha_k (d_k - t), 0 \}.$$

This is true because the graphs underneath immediately show that $c''_k(t) = -c'_k(t) + \beta_k$.



We see that, by choosing β_k and d in this way, we have effectively maximized $\sum \alpha_k E_k$, the weighted sum of earlinesses, so that we have solved the $n|1||\sum \alpha_k E_k$ problem! Notice that $\sum \alpha_k E_k$ is not a regular measure.

We shall return to the formulation of Lawler and Moore in 4.2., when we consider the $n|1||\sum \alpha_k T_k$ problem.

3.5. Branch-and-bound methods (BB)

One of the most promising techniques for solving optimization problems is the technique called "branch-and-bound". Originally developed by Land and Doig in the context of integer programming, it is being applied to a growing number of problems such as non-linear programming, the quadratic assignment problem and the travelling salesman problem, where it was used in the classic paper by Little, Murty, Sweeney and Karel [62]. A fairly recent survey is given by Lawler and Wood [58].

A general description of a branch-and-bound algorithm, shall be given now. In general, the set of all possible solutions to the minimization problem is being split up stepwise in disjunct subsets. For each subset a lower "bound" is calculated: the value of the objective function for each solution in the subset will be larger than or at least equal to this lower bound. We then choose a subset from where we can "branch"; this could be the one with the presently lowest lower bound, but an other way to choose is possible and will be mentioned later on. "Branching" now implies further splitting up the subset in disjunct parts. As soon as one of these subsets contains only one element, we have a complete solution for which we can calculate the value V of the objective function. We can from then onwards disregard all subsets with a lower bound greater than V ; no improvement can be found in them. We continue the branching and bounding, continuously comparing lower bounds with the present best complete solution, until we have a complete solution whose value is smaller than or at least equal to all remaining lower bounds. This solution is the desired optimum one.

We see then that a branch-and-bound algorithm is determined by three prescriptions:

- (1) the bounding prescription, i.e. how to calculate a lower bound;
- (2) the branching prescription, i.e. how to split up a subset of solutions;
- (3) the searching prescription, i.e. how to choose a new branching point.

Now (1) obviously is very important and the quality of any branch-and-bound algorithm is mainly determined by the sharpness of the bounds.

Rules with regards to (2) are often incomplete in the sense that they do not uniquely determine how to split up the subset under consideration. Various heuristic rules may then be employed to arrive at the definite splitting.

Finally, (3) is sometimes not explicitly given in literature, and is mainly an administrative matter. Basically one can distinguish two different approaches:

- (3a) branch from the subset with the present lowest bound ("frontier search");
- (3b) branch from the most recently created subset ("newest active node"*) .

Method (3b) usually leads to more branching operations than (3a), but requires little computer storage (of the pushdown-stack type), whereas (3a) demands large space for the storage of intermediate data.

*) One could combine the two by branching from that subset among the most recently created ones, that has the lowest bound.

Branch-and-bound methods have been very successful in solving sequencing problems in general and some machine scheduling problems in particular. Various examples will be dealt with in other sections; especially in the $n|m|G$ problem branch-and-bound methods have been used extensively. However, a recent article by Bratley, Florian and Robillard [34] indicates that already a $10|10|G$ problem poses great problems and can probably not be solved solely by branch-and-bound methods.

Nevertheless, branch-and-bound methods have heuristic value as well; if one is willing to be satisfied with a solution within, say, 10% of the optimum, and a complete solution with value V is known, all subsets with lower bounds greater than $11V/10$; this should speed up calculations considerably.

We purposely refrain from going any specific example at this point as we have done in other sections. As mentioned before, applications of branch-and-bound methods are so numerous throughout this report that they will sufficiently illustrate the power of this method.

3.6. Combinatorial-analytical methods (CA)

By combinatorial-analytical methods we mean all theoretically derived results whereby either the set FS of feasible solutions is effectively reduced to a much smaller set POS of potentially optimal ones, or a constructional method to find the optimum is explicitly given.

In the first case, results usually have the form: "if a sequence has property P , this sentence can never be optimal", "there exists an optimal sequence with property P " or "any optimal sequence has property P ". The third formulation is much stronger than the second one: propositions of the second type are not

necessarily "additive", by which is meant that, if we have a number of these propositions referring to properties P_1, \dots, P_n , this does not imply that there is one optimal sequence which has all of these properties.

In the second case, the problem is, of course, solved: we have a constructional method that enables us to find an optimal sequence. Results like this are, however, comparatively rare in machine scheduling theory and they are generally confined to very simple situations, such as the $n|2|F|F_{\max}$ problem, solved by Johnson's classic paper [54] in 1954.

Many examples of these results will be given throughout this report, so again there is no need to go into details here. Nevertheless, it would be nice if one could give a few generally applicable results here. The theory of combinatorial optimization, however, has hardly been developed so far and the only interesting theorem was given by Smith [93] in 1956:

Theorem 3.6.A: a sufficient condition that $f(\bar{\pi}) \leq f(\pi)$ for all π , where f is a real function defined on permutations π of $(1, \dots, n)$ is that:

- (1) there exists a function g , defined on ordered pairs (k, ℓ) such that, if

$$\pi = (i_1, \dots, i_k, i_{k+1}, \dots, i_n) \text{ and}$$

$$\pi' = (i_1, \dots, i_{k+1}, i_k, \dots, i_n), \text{ then}$$

$$f(\pi) \leq f(\pi') \text{ if } g(i_k, i_{k+1}) \leq g(i_{k+1}, i_k);$$
- (2) $\bar{\pi}$ is such that k precedes ℓ if $g(k, \ell) \leq g(\ell, k)$.

Proof: in any $\pi \neq \bar{\pi}$, we can interchange the pair (i_k, i_ℓ) , where i_ℓ immediately precedes i_k in π , but follows i_k in $\bar{\pi}$. By (2),

$g(i_k, i_\ell) < g(i_\ell, i_k)$, so by (1) the interchange does not increase $f(\pi)$.

The situation is even simpler when g is function of one variable only, k preceding ℓ , if $g(k) < g(\ell)$. In this case, g is necessarily transitive; in the general case, if $g(k, \ell) < g(\ell, k)$ and $g(\ell, m) < g(m, \ell)$, it does not necessarily follow that $g(k, m) < g(m, k)$. So one has to check if a sequence (i_1, \dots, i_n) where k precedes ℓ if $g(k, \ell) < g(\ell, k)$ can be constructed at all.

No general constructional method for g is given, but in general one interchanges elements i_k and i_{k+1} and tries to write the resulting change in the value of f as a function of these two elements only. A more abstract formulation of this idea is given by Elmaghraby [29].

Examples of this method will be given in chapters 4 and 5; by the nature of theorem 3.6.A. applications are restricted to those cases where the value of the optimality criterium is determined by one permutation only.

As announced, we shall not give any specific examples here. It is interesting to point out, however, that the usefulness of theorem 3.6.A. is due to the fact that it permits one to find an optimum by only checking the effect of interchanging pairs of elements. The theorem guarantees that our local optimum (in the sense of Nicholson [79]), is also global.

3.7. Algebraic methods (A)

There have been only a few attempts to solve scheduling problems by algebraic methods. By the latter we mean those methods that concentrate on structural properties of the set of all operations, and on the relations between them. Here we shall pay attention to the work of Giffler ([37], [38]) on schedule algebras and the work of Rial ([83]) and Driscoll and Suyemoto ([26]) on relation nets. The lack of any further research in this direction explains why we shall see no need to return to these methods any more after this section.

We realize that Giffler's approach is aimed at situations lacking the characteristic difficulty of machine scheduling problems: in his schedule algebra theory, it is assumed that a complete ordering of jobs is (implicitly or explicitly) given, in which case the schedule graph is equal to a PERT-CPN type of network. Also we realize that Rial's approach is aimed at far more general problems than the machine scheduling problem. However, we think it not unlikely that algebraic methods may turn out be powerful aids in solving this problem and therefore describe the two approaches in somewhat more detail.

3.7.1. Schedule algebras

In schedule algebra theory, we generally try to solve the well-known problem: given n strictly-ordered activities and the starting times of all unpreceded ones, what is the earliest starting time of each activity?

We assemble all relevant information in a $n \times n$ -matrix S , with

$$s_{ij} = \begin{cases} \{t_{ij}\} & \text{if } i \ll j \\ 0 & \text{otherwise,} \end{cases}$$

where $\{t_{ij}\}$ is the set of minimum intervals between the start of activity i and the start of activity j , arising from various technological and other considerations^{*)}. If t_{ij} has "zero magnitude", we shall denote this by \emptyset , to avoid confusion with $s_{ij} = 0$, which indicates that i does not directly precede j .

We shall now study the structure of all matrices of this type whose essential characteristic is that its elements are sets of real numbers (including \emptyset), or 0 (zero). We can define two relevant ways to add and multiply these matrices. For the first way define $C = A \oplus B$ where A and B are both $(n \times m)$ -matrices by defining $c_{ij} = a_{ij} \oplus b_{ij}$ by the following procedure:

- (1) collect all entries of the sets a_{ij} and b_{ij} ;
- (2) replace by zero all combinations with the same magnitude, but different signs;
- (3) if all entries are now zero, suppress all but one; if not, suppress all zero's.

Multiplication is then defined as follows: $D = A \odot B$ where A is a $(n \times m)$ -matrix and B a $(m \times p)$ -matrix, has as i - j th entry

$$d_{ij} = (a_{i1} \odot b_{1i}) \oplus \dots \oplus (a_{im} \odot b_{mj});$$

to define $a_{ik} \odot b_{kj}$, we take all pairs of elements $(\alpha_{ik}, \beta_{kj})$ from both sets, form

$$\alpha_{ik} \odot \beta_{kj} = \begin{cases} |\alpha_{ik}| + |\beta_{kj}| & \dots\dots\dots \text{if they have the same sign} \\ -|\alpha_{ik}| - |\beta_{kj}| & \dots\dots\dots \text{if they have different signs} \\ 0 & \dots\dots\dots \text{if one or both are zero} \end{cases}$$

^{*)} We use the term "activities" instead of "operations".

and add all these products, according to the above definition of \oplus .

Under this addition, \oplus , the $(n \times m)$ -matrices form an additive group with the matrix that is identically zero as neutral element. The $(n \times n)$ -matrices form a non-commutative ring with identity matrix I , that has $\{1\}$ on the diagonal and zero elsewhere. As with real matrices, inverses according to multiplication are unique (if they exist).

For the second way to define addition and multiplication, we remark that, whereas the above operations shall turn out to produce the time-length of all possible paths between two activities, usually we are only interested in the maximum length of these paths. So we restrict ourselves to situations where all matrices have entries that are either i , a positive real number or zero and define $E = A * B$ by

$$e_{ij} = \max \{a_{ij}, b_{ij}\},$$

treating 0 as negative infinity; and $F = A \# B$ by

$$f_{ij} = \max_k \{(a_{ik} \oplus b_{kj})\}.$$

The reason that we did not immediately introduce these definitions, is that the set of all these matrices (where now we just as well replace the set a_{ij} , that is the i - j th element, by $\max a_{ij}$) has much less structure under these definitions; they do not even form a group any more.

Returning now to the previously defined matrix S , that in fact gives the length of all "one-level chains", $i \ll j$, we see that $S \oplus S = S^2$ effectively gives the length of all two-level chains $i \ll k \ll j$, the i - j th entry being

$$\{t_{ik} + t_{kj}\},$$

in so far as t_{ik} and t_{kj} are not zero. Analogously, the set that is the i - j th element of S^w gives the lengths of all possible w -level chains. Obvious S^w will be identically zero if $w > \lambda$ for some λ . Defining Θ as follows:

$$\Theta = I \oplus S \oplus S^2 \dots \oplus S^\lambda \quad (22)$$

θ_{ij} gives the lengths of all chains from i to j .

It is now easy to prove that

$$\Theta^{-1} = (I \oplus (-I \otimes S))^{-1} \quad (23)$$

where $-I$ has -1 on the diagonal and zero elsewhere. To do so, we multiply both sides of (23) by $(I \oplus (-I \otimes S))$, getting:

$$\Theta \oplus (-I \otimes S \otimes \Theta) = I$$

or

$$\Theta = I \oplus (S \otimes \Theta)$$

which follows directly from (22), because $S^{\lambda+1} \otimes \Theta$ is identically zero. Elsewhere [38], Giffler gives efficient methods to determine the inverse of a schedule matrix.

Now, if we are only interested in the maximum length of all chains from i to j , we compute:

$$\Phi = I * S * (S \# S) * (S \# S \# S) * \dots * \underbrace{(S \# \dots \# S)}_{\lambda}$$

the i - j th entry of Φ giving the desired information.

Given a $(1 \times n)$ vector T , where

$$T_j = \begin{cases} t_j, & \text{the earliest starting time of activity } j, \\ & \text{if } j \text{ is unpreceded} \\ 0 & \text{otherwise} \end{cases}$$

we compute

$$\bar{T} = T \# \bar{\Phi}$$

which gives the earliest possible starting time for all n activities.

Writing $S \# S$ as $S^{\#2}$, etc., we have

$$\begin{aligned} \bar{T} &= T \# (I * S * S^{\#2} * \dots * S^{\#\lambda}) \\ &= (T \# I) * (T \# S) * ((T \# S) \# S) * \\ &\quad ((T \# S^{\#2}) \# S) * \dots * ((T \# S^{\#\lambda-1}) \# S) \end{aligned}$$

which gives rise to the recursive formula:

$$\bar{T} = T_\lambda$$

where:

$$T_0 = T$$

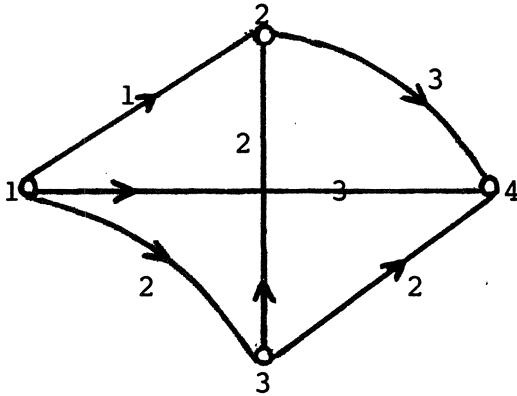
$$T_k = T_{k-1} * (T_{k-1} \# S), \quad k = 1, 2, 3, 4, \dots$$

($T_\lambda = T_{\lambda+r}$, $r = 1, 2, \dots$, because $A * A = A$ for all A).

Example: suppose we have

$$S = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 3 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

which corresponds to the following graph of activities:



We find:

$$S^2 = \begin{pmatrix} 0 & 4 & 0 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$S^3 = \begin{pmatrix} 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$S^4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\theta = \begin{pmatrix} 1 & 4 & 2 & 7 \\ 0 & 1 & 0 & 3 \\ 0 & 2 & 1 & 5 \\ 0 & 0 & 8 & 1 \end{pmatrix}$$

If $T = (3 \ 0 \ 0 \ 0)$, we find:

$$\begin{aligned} T_1 &= T * (T \# S) \\ &= (3 \ 4 \ 5 \ 6) \end{aligned}$$

$$\begin{aligned} T_2 &= T_1 * (T_1 \# S) \\ &= (3 \ 7 \ 5 \ 7) \end{aligned}$$

$$\begin{aligned} T_3 &= T_2 * (T_2 \# S) \\ &= (3 \ 7 \ 5 \ 10) \end{aligned}$$

$$\begin{aligned} T_4 &= T_3 * (T_3 \# S) \\ &= (3 \ 7 \ 5 \ 10) \end{aligned}$$

so $\bar{T} = (3 \ 7 \ 5 \ 10)$.

The method of schedule algebras can be extended to the situation where the directly-precedes relations are given implicitly by some priority rule (such as First On, First Off, etc.). In its present form, it can, however, not contribute directly towards the solution of the machine scheduling problem, because the fundamental relation: "i precedes j or j precedes i" cannot be expressed^{*)}. For an approach, where these (and many other) relations are readily available, we turn to so-called relation algebras.

3.7.2. Relation algebras

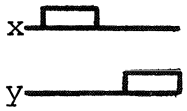
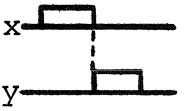
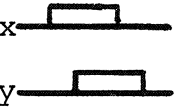
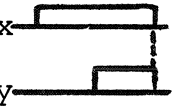
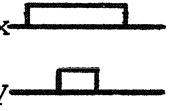
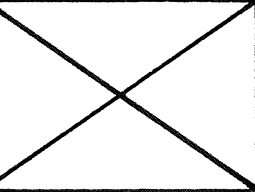
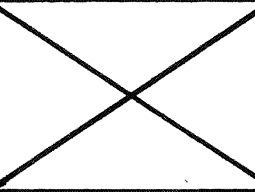
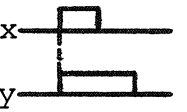
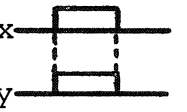
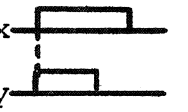
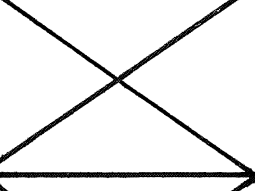
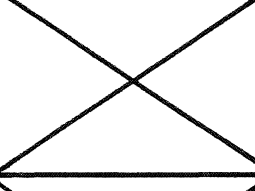
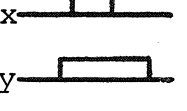
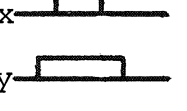
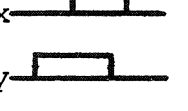
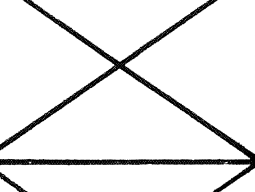
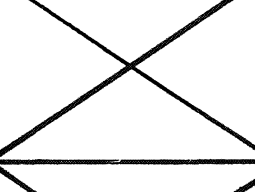
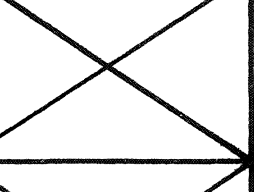
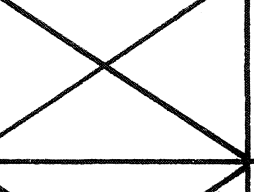
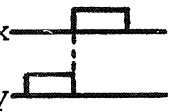
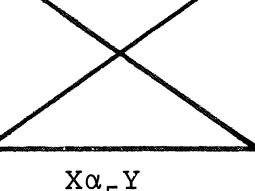
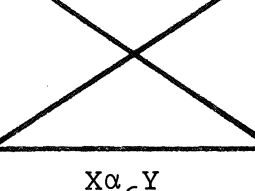
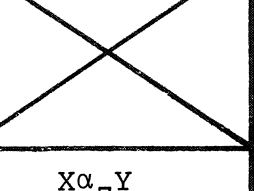
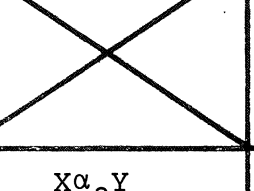
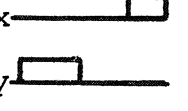
The basic idea of relation algebra, as presented somewhat forbiddingly by Rial [83], and Driscoll and Suyemoto [26], is

*) Schedule algebras can, of course, be used as part of a general algorithm to solve a $n|m|G$ problem, (see, for instance, Ashour and Parker []).

the following. Suppose we have a set of n activities with all kinds of time-relations between them, either very vague (i.e., X starts before Y) or very precise (Y begins exactly when X stops, X starts n time-units after Y). Especially in large projects these relations may well lead to logical contradictions. We want to discover these contradictions (if they exist) and find out how they can be dissolved.

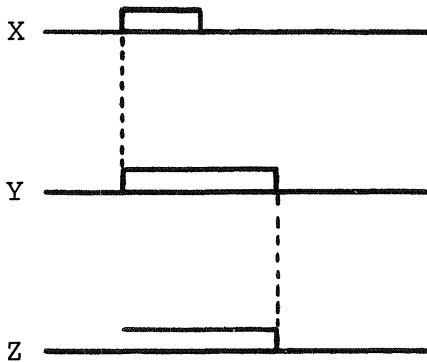
First then, we have to classify all possible relations. Now each activity X is characterized by its starting time t_x and finishing time T_x . Likewise, Y is characterized by t_y and T_y . There are five possible relations between t_x , and t_y and T_y : $t_x < t_y$, $t_x = t_y$, $t_y < t_x < T_y$, $t_x = T_y$, $t_x > T_y$; the same relations exist between T_x , and t_y and T_y . Of the 25 resulting combinations, 12 turn out to be infeasible, which leaves 13 fundamental relations. They are illustrated by the scheme below^{*)}:

*) Our notation differs from Rial's.

	$T_x < t_y$	$T_x = t_y$	$t_y < T_x < T_y$	$T_x = T_y$	$T_x > T_y$	
$t_x < t_y$						$X\alpha_0 Y$
$t_x = t_y$						$X\alpha_1 Y$
$t_y < t_x < T_y$						$X\alpha_2 Y$
$t_x = T_y$						$X\alpha_3 Y$
$t_x > T_y$						$X\alpha_4 Y$
	$X\alpha_5 Y$	$X\alpha_6 Y$	$X\alpha_7 Y$	$X\alpha_8 Y$	$X\alpha_9 Y$	

Each feasible combination is illustrated by two time scales with the positions of X and Y. As to notations, $X\alpha_{ij}Y$ has to be read as: $(X\alpha_i Y) \wedge (X\alpha_j Y)$, \wedge standing for logical conjunction.

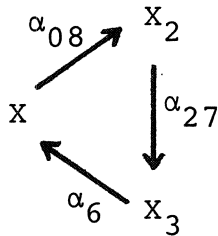
In order to discover logical inconsistencies, we introduce the concept of implication: if it follows from $(X\beta Y) \wedge (Y\gamma Z)$, that then $(X\delta_1 Z) \vee \dots \vee (X\delta_n Z)$, the relation $\delta_1 \vee \delta_2 \vee \dots \vee \delta_n$ is said to be the implication (or the product) of β and γ ; \vee is the sign for logical disjunction. By example, if $X\alpha_{17}Y$ and $X\alpha_\delta Z$, then $X(\alpha_5 \vee \alpha_6 \vee \alpha_7)Z$, as will be obvious from the picture below.



We can extend these relations by defining $X\hat{\beta}Y$ to mean $Y\beta X$ (i.e., $\hat{\alpha}_{05} = \alpha_{49}$) and by defining $X\bar{\beta}X$ to mean that $X\beta Y$ is not the case. All implications and conjunctions have extensively been tabularized by Driscoll and Suyemoto.

Now, if there is any logical inconsistency in the network of relations, it will necessarily arise out of some loop $X_1\beta_1 X_2\beta_2 X_3 \dots \beta_{n-1} X_n \beta_n X_1$. To discover this, we transform each of these loops step by step by means of implications into a relation of the type $X_1\gamma_1 X_1\gamma_2 X_1$, which is identical to $X_1(\gamma_1 \wedge \hat{\gamma}_2)X_2$. We then check in a table of conjunctions if this conjunction is false (i.e.: no pair (X_1, X_2) could possibly have this relation). If so, we have an inconsistency.

Example: suppose we have the following cycle:



$X_1 \alpha_{08} X_2$ and $X_2 \alpha_{27} X_3$ imply $X_1 \alpha_7 X_3$. Now $X_1 (\alpha_7 \wedge \hat{\alpha}_6) X_3$ is false, because $X_1 \alpha_7 X_3$ means $t_{x_3} < T_{x_1} < T_{x_3}$ and $X_1 \hat{\alpha}_6 X_3$ means $T_{x_3} = t_{x_1} (< T_{x_1})$.

An obviously indispensable result which we need here, is:

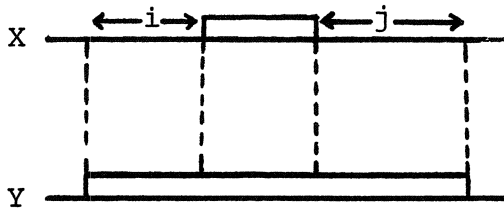
Theorem 3.7.A: $X_1 \beta_1 X_2 \dots \beta_{n-1} X_n \beta_n X_1$ and

$X_k \beta_k X_{k+1} \dots \beta_{n-1} X_n \beta_n X_1 \beta_1 X_2 \dots \beta_{k-1} X_k$ have the same truth value for $k = 2, \dots, n$.

Proof: trivial for $n = 2, 3$; from there by induction.

Given the network of relations and the tables, the search for inconsistencies, described above, can easily be carried out by a computer. Rial announces a program in preparation; no results have been presented since then. Driscoll and Suyemoto present a number of heuristic rules whereby a logical conflict might be solved.

Rial has extended his approach to so-called metrized relations, where not only is given that, for instance, $t_x < t_y$, but where we know that $t_y = t_x + i$. The notation is easily extended, to cover these relations, the above example being written as $X \alpha_0(i) Y$, and, for instance, $X \alpha_{27}(i, j) Y$ denoting the following situation:



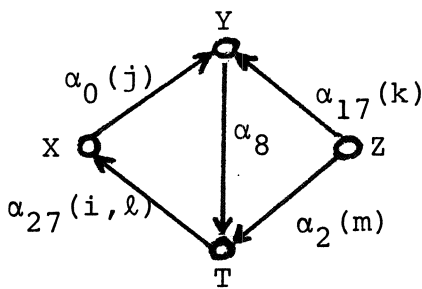
In the case of metrized relations we can again study the implications of two metrized relations $\beta_1(i)$ and $\beta_2(j)$. However, what is more important is that metrized relation place a number of restrictions (in the form of linear equations) on the parameters i, j, \dots and the durations $d_x = T_x - t_x$ of the activities. For instance, in the above example, we have

$$i + d_x + j = d_y$$

which must be true if the relation $X\alpha_{27}(i,j)Y$ is true.

In this way, a number of necessarily valid equations can be derived from a true metrized relation network. Let us illustrate what we can do with them by a final example.

Suppose we have the following network (one can think of T as a common time base).



The network can be shown to be true in a logical sense. We take all cycles and derive equations from them (tables exist for this procedure).

$$X\alpha_0(j)Y\alpha_8T\alpha_{27}(i, l)X : i + d_x + l = d_T$$

$$d_x + l = j + d_y$$

$$Y\alpha_{17}(k)Z\alpha_2(m)T\hat{\alpha}_8^Y : d_Y = d_Z + k$$

$$m + d_Z + k = d_T$$

$X\alpha_0(j)Y\alpha_{17}(k)Z\alpha_2(m)T\alpha_{27}(i,\ell)X$ gives no new information.

So we have:

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \\ d_z \\ i \\ j \\ k \\ \ell \\ m \end{pmatrix} = \begin{pmatrix} d_T \\ 0 \\ 0 \\ d_T \end{pmatrix}$$

(d_T is assumed constant).

Now if we want to know the effect of a small change in the variables (especially the influences these changes have on each other), we know that the augmented variables must satisfy the same equations, and get by subtracting:

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \Delta d_x \\ \Delta d_y \\ \Delta d_z \\ \Delta i \\ \Delta j \\ \Delta k \\ \Delta \ell \\ \Delta m \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

By elementary row transformations we find that the matrix of coefficients is equivalent to

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \end{pmatrix}$$

which implies that we can choose Δj , Δk , Δl and Δm , and then solve for Δd_x , Δd_y , Δd_z and Δi . The "conditional conflict" (as Rial calls it) has been averted.

It cannot be denied that the examples given are extremely artificial. Nevertheless, the algebraic methods at least fully employ the structural properties of the scheduling problem, however, inelegant they may seem. There is room for improvement here, and subsequent developments may well justify the attention paid to the methods here.

3.8. Sampling techniques (ST)

In this section we enter the realm of heuristic methods, by which we shall generally mean methods that cannot strictly guarantee the finding of an optimum solution.

By far the most important heuristic methods are those that use more or less sophisticated priority rules. Designing these rules and comparing their performance by extensive simulation has kept many researchers happy and busy. We shall present the main results in the next section, but here we want to pay attention to a curious feature of the machine scheduling problem, that has been exploited by Heller [49] and others.

The background of their methods is that the number of distinct maximum flow times N_d is relatively small, especially in the

$n|m|F$ or $n|m|P$ situation. For a $10|5$ situation, the number of possible sequences is 6.29×10^{32} , whereas N_d is 9.38×10^{11} in the flow-shop situation and 1.13×10^{15} in the job shop situation^{*)}. In an $n|m|P$ situation, the maximum flow time is a sum of $(n + m - 1)$ processing times, which gives an immediate upper bound on N_d of $\binom{nm}{n + m - 1}$. This indicates that it might be profitable to study the distribution of the different times over the population of all possible schedules. Heller has conducted some experiments in this direction, and has concluded (and derived theoretically) that this distribution is asymptotically normal.

The practical use of Heller's work is not at all clear. One is, of course, mainly interested in what happens round the lower tail of the normal distribution, where the fit is worst. Moreover, if one wants to simulate a great number of different solutions, there are more efficient populations to sample from than the population of all feasible schedules. There is an application, cited enthusiastically by Elmaghraby [29], which boils down to fitting a normal distribution to the results gained so far and calculating therefrom the probability of finding a better schedule than the present best one in the next simulation. Surely this process rests on very weak theoretical grounds; not surprisingly, practical applications have not been reported so far.

3.9. Heuristic methods (H)

By now it will have become apparent that an optimum solution to a scheduling problem is generally not so easy to find. Taking into account as well that it is already difficult enough

*) Reported by Ashour [].

to isolate a scheduling problem from a host of surrounding complex problems, it is altogether not surprising that only a few practical applications of pure scheduling theory are known. What happens in most cases is that, given a particular scheduling problem, one tries to develop a method that will generally produce "good" sequences, although it cannot guarantee to find an optimum one. These "suboptimal" methods we shall call heuristic. We shall deal with them here and for the rest of the report stick to methods that really guarantee optimal solutions.

Research into heuristic methods has mainly concentrated on testing different kinds of so-called priority rules. Generally, the technique of testing any heuristic method is to use that heuristic method to generate one (or more) feasible schedule(s) for a given problem. Then one evaluates the quality of the (best) schedule, and repeats the whole experiment with either the same data and a different method (so as to compare methods) or different data and the same method (so as to get an impression of the quality of the method in general).

Now a schedule is completely determined if the starting-times of all operations are known. If the schedule is generated in such a way, that a decision taken with regard to the starting-time of any particular operation can never be revoked, the procedure is called a single-pass one. The fact that almost all known procedures are single-pass ones is a serious limitation, as most human beings, operating for instance on a Gantt chart, continuously change previous decisions. More research on simulation of this adjusting behaviour is badly needed*).

*) The only available study is by Dutton ([28]).

If in a single-pass procedure decisions are taken "on the spot" (which means that they can be taken in the order in which they are implemented), we speak of a dispatching procedure. Again, most known methods belong to this class.

We now introduce the important concept of the set S_o of schedulable operations. At any time this is the set of all operations whose predecessors have all been scheduled. It therefore consists of exactly n operations, one for each job. Scheduling one of these operations implies moving it to the set S_p of the (m) operations in progress. S_o can be split up:

$$S_o = S_o^1 \cup \dots \cup S_o^m,$$

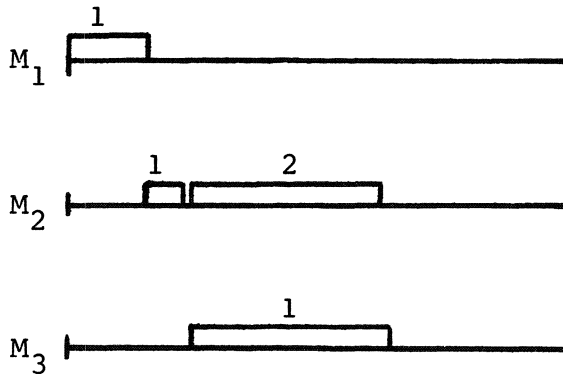
where S_o^l contains all operations to be scheduled at machine l .

Now, if C_l is the finishing-time of the present operation on machine l and s_{kl} is the potential starting-time of $(k,l) \in S_o^l$, then the earliest possible start and finish-times of (k,l) are given by $\max(C_l, s_{kl})$, resp. $\max(C_l, s_{kl}) + p_{kl}$.

If we choose as the next operation to be scheduled any one with minimal earliest possible starting-time, we get a so-called non-delay schedule; similarly, if we choose any one that starts before the minimal earliest possible finishing-time, we get a so-called active schedule.

In general, an active schedule is one where it is not possible to decrease the starting-time of any operation without increasing the starting-time of another one (Conway, Maxwell and Miller [24], page 111). Obviously, any optimal schedule must be active. A non-delay schedule is an active schedule where at no time a machine stands idle on which a schedulable operation could have been processed. An optimal schedule, however, is not

necessarily non-delay. Take, for example, the optimal sequence for a $2|3|F|F_{\max}$ problem where $p_{21} = p_{23} = 0$, that is illustrated below and that has a delay on M_2 :



By randomly breaking ties, we can generate a number of active and non-delay schedules and compare their performance^{*)}. This has been done by Bakhru and Rao (reported in [24]) and leads to the general conclusion that non-delay schedules behave better in general. However, things get more realistic if ties are not broken randomly, but by application of some priority rule, or if - alternatively - an operation is selected from S_0 by this priority rule and the starting-time is then so determined as to produce an active or non-delay schedule. Many of these priority rules have been developed and tested (Day and Hottenstein [25], Gere [36]). To name but a few, one can grant highest priority to the operation (k, ℓ) where

- (1) J_k has the earliest due-date;
- (2) J_k has either highest or lowest slack-time (i.e. difference between time remaining before the due-date and sum of remaining processing times);

*) This is similar to the approach by Giffler and Thompson [39].

- (3) J_k has lowest slack-time per remaining operation;
- (4) (k, ℓ) arrived first in S_O^ℓ (FCFS: first come, first served, or FIFO: first in, first out);
- (5) J_k has lowest shop arrival time r_k ;
- (6) $p_{k\ell}$ is minimal (SPT: shortest processing time, or SOT: shortest operation time);
- (7) J_k has either minimal or maximal total remaining processing time;
- (8) J_k has minimal total processing time;
- (9) J_k has either minimal or maximal number of remaining operations;
- (10) (k, ℓ) has minimal set-up time;
- (11) (k, ℓ) is chosen in a completely random manner.

Other priority rules can be found in the literature mentioned above; Day and Hottenstein [25] give many references. The performance of most of these rules has been extensively investigated. We cite Conway, Maxwell and Miller [24], who report a study by Jeremiah, Lalchandani and Schrage, which proved among other things that priority rules work best in combination with non-delay schedules, that SPT scheduling and random scheduling (sic) are about equally superior on active schedules, and that the "maximum remaining work load" criterium performs reasonably well on the whole. However, there is no obviously "best" rule. The latter remark coincides reasonably well with the results of Gere [36]. He finds that rules based

on jobs slack are slightly better than SPT scheduling, which is in turn slightly superior to the equally bad random and FCFS-method.

Next, however, Gere moves on to add some additional heuristic rules, two of which turn out to be very effective: an "alternate operation" rule, whereby job l is chosen instead of the originally picked job k , if the choice of k threatens to cause overdue delivery of job l , and a "look ahead" rule, which forces the chosen job k to wait if a more critical job is on its way. He conjectures that all previously tested procedures will work about equally well when bolstered by these two additional rules, but does not present any definite evidence. His conclusion is nevertheless that the choice of additional heuristics is far more important than the choice of a priority rule itself. One might therefore just as well choose the easiest one available (SPT). All together, these heuristic methods are (not surprisingly) superior to Heller's sampling approach, reported in 3.8.

A more sophisticated development, also reported in [24], are methods whereby one varies between using one priority rule and the completely random method by assigning non-equal probabilities to each operation in S_0 , the job with the highest priority getting the highest probability. Again, the results are not consistently better than either of the two extremes, but a surprising outcome of some experiments (by Nugent) is that, with some procedures, there is a certain degree of randomness that is clearly superior to both complete randomness and complete determinacy. The reasons for this amply demonstrated fact are not clear.

Concluding this section we feel that in general heuristic methods have not been sufficiently explored and have been

interpreted too narrowly. More work should be done on heuristic methods that are tailor-made for a particular problem (e.g., Burstall [20]), and more attention should be devoted to simulating the methods of a good human scheduler. It is not unlikely that, given the present poor state of applicable scheduling theory, good heuristic methods will continue to be of utmost practical importance.

3.10. Conclusion

In this chapter we reviewed existing methods to attack the machine scheduling. Most of them typically try to eliminate sequences that are obviously non-optimal. (A method like complete enumeration which does not do this, may be rejected straight away). This elimination is performed in various ways: branch-and-bound methods try to evaluate the quality of a partly filled schedule as early as possible, dynamic programming always chooses the best of equivalent partly filled schedules to proceed with, combinatorial-analytical techniques rely on careful judgment of the effect of certain interchanges in a sequence. These methods are in fact the best we have at the moment. As stated we do not believe integer programming will ever produce an optimal solution method to the scheduling problem, nor do we have much faith in Heller's sampling method. Algebraic and heuristic methods deserve more attention, the latter ones probably dominating in real-life situations for many years to come.

4. Some special cases

4.1. Introduction

In this chapter the techniques described in chapter 3, shall be applied to a few special and (comparatively) simple machine scheduling problems. Most prominent among them is the $n|1$ problem, on which a lot of work has been done. Still, even here many problems remain to be solved. We devote special subsections to situations where there are additional precedence constraints among the jobs. Furthermore, we pay attention to the situation where instead of one machine we have m identical machines to perform the jobs on.

The two-machines and three-machines problem also deserve some special attention; Johnson's work on the $n|2|F|F_{\max}$ problem in 1954 aroused new interest in machine scheduling problems in general. Finally we pay attention to the $2|m$ situation, mainly because of the interesting graphical method designed to solve problems there.

4.2. The one-machine problem ($n|1$)

Most theoretical work on machine scheduling problems pertains to the $n|1$ situation. We shall try to give a review of known results, classifying them by the various optimality criteria in a way analogous to 2.3..

There are a few remarks to be made beforehand. Firstly, it is trivial to prove that in solving a $n|1||\phi(c_1, \dots, c_n)$ problem, where $\phi(c_1, \dots, c_n)$ is a regular measure of performance, one does not have to consider any schedule with job splitting or idle time. In both cases the schedule could be improved in an obvious way.

Secondly, it is clear that well-known criteria like F_{\max} , C_{\max} and W_{\min} are now independent of sequence and do not have to be considered.

In view of the first remark we only have to consider the $n!$ different permutation schedules. As to notations, we denote by i_k or j_k the job number that in a given permutation occupies the k^{th} place. For example, $i_8 = 2$ means job 2 is in the eighth position. Furthermore, we can write p_k for p_{k1} and W_k for W_{k1} ($k = 1, \dots, n$).

4.2.1. Criteria based on completion-dates and flow-times

Having assumed that $r_k = 0$ for all k , important criteria to consider here are

$$\bar{F} \text{ and } \Sigma \alpha_k F_k$$

the former one being equivalent to \bar{W} and \bar{L} , the latter one to $\Sigma \alpha_k W_k$ and $\Sigma \alpha_k L_k$.

The $n|1||\bar{F}$ problem is easily solved and the solution has been known for a long time. Denoting a sequence by i_1, i_2, \dots, i_n , we find

Theorem 4.2.1.A: the $n|1||\bar{F}$ problem is solved by the sequence i_1, \dots, i_n with

$$p_{i_1} \leq p_{i_2} \leq \dots \leq p_{i_n}$$

Proof: $F_{i_k} = \sum_{j=1}^k p_{i_j}$, so $\bar{F} = \frac{1}{n} \sum_{j=1}^n (n - i + 1) p_{i_j}$;

this sum is minimized by arranging the $p_{[i]}$ in order of increasing magnitude. A graphical "proof" is also given in Conway, Maxwell and Miller [24].

This way of sequencing is called: shortest-processing-time sequencing (SPT). It also minimizes \bar{W} , \bar{L} (and \bar{C}), W_{\max} , C_{\min} and $\frac{1}{n} \Sigma F_k^\alpha$ ($\alpha > 0$). To prove the latter one notes that is

$p_{i_k} > p_{i_{k+1}}$ in some sequence, one can interchange these two jobs, thereby holding $F_{i_{k+1}}^\alpha$ constant and increasing $F_{i_k}^\alpha$ *).

The $n|1||\Sigma\alpha_k F_k$ problem is hardly more difficult to solve.

Theorem 4.2.B: the $n|1||\Sigma\alpha_k F_k$ problem is solved by the sequence i_1, \dots, i_n with

$$p_{i_1}/\alpha_{i_1} < p_{i_2}/\alpha_{i_2} < \dots < p_{i_n}/\alpha_{i_n}$$

Proof (Smith [93]): given a sequence i_1, \dots, i_n , and interchanging i_k and i_{k+1} , the old sequence will be better than or as good as the new one if

$$\alpha_{i_k} \sum_{j=1}^k p_{i_j} + \alpha_{i_{k+1}} \sum_{j=1}^{k+1} p_{i_j} < \alpha_{i_{k+1}} \sum_{j=1}^{k-1} p_{i_j} + p_{i_{k+1}} + \alpha_{i_k} \sum_{j=1}^{k+1} p_{i_j}$$

or

$$\alpha_{i_{k+1}} p_{i_k} < \alpha_{i_k} p_{i_{k+1}}$$

$$p_{i_k}/\alpha_{i_k} < p_{i_{k+1}}/\alpha_{i_{k+1}}$$

We have found a function $g(k)$ as described in theorem 3.6.A.; the proof is now immediate.

4.2.1.1. Precedence constraints

We now turn to the more complicated situation where there are precedence constraints among the jobs. (dropping assumption (J6)). We can represent these constraints by a directed graph, nodes representing jobs and a directed arc linking J_k with $J_{k'}$, implying that J_k should precede $J_{k'}$.

*) We really use here (as below) theorem 3.6.A..

Let us, however, first treat the simple case, where has been split up in groups G_i of n_i jobs, where each group has to be executed consecutively in a given order^{*)}. We then have, if $J_k \in G_i$:

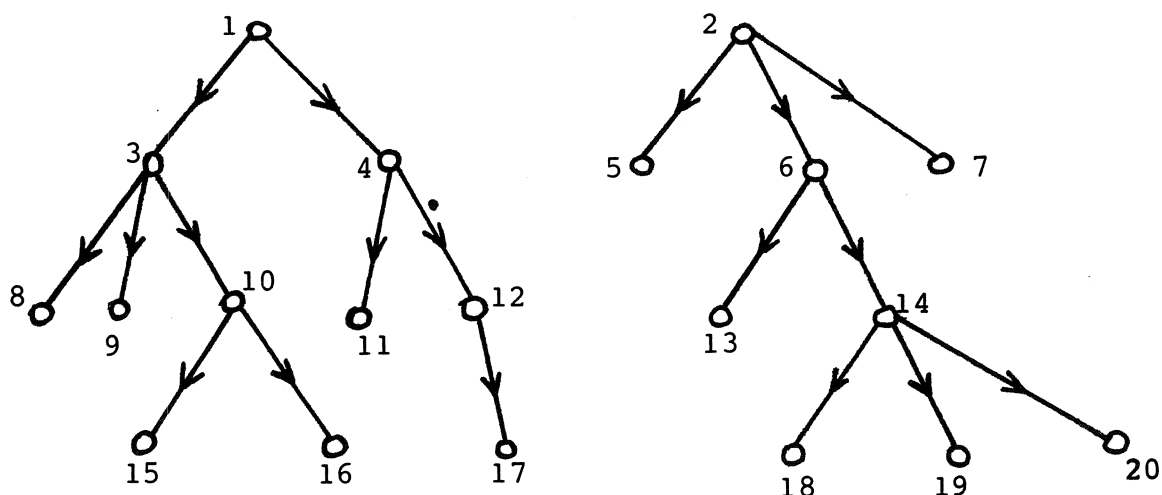
$$F_k = F_{G_i} - c_k$$

where F_{G_i} is the flow time of G_i and c_k is a constant, equal to the sum of the processing times of the jobs following J_k in G_i . Then:

$$\sum \alpha_k F_k = \sum_{G_i} (\sum \alpha_k) F_{G_i} - \sum \alpha_k c_k$$

and from theorem 4.2.B. we see that the optimal sequence of the groups is given by ordering them according to increasing $(\sum p_k) / (\sum \alpha_k)$ ratio. This solves this particular $n|1|(J6)|\sum \alpha_k F_k$ problem, and therefore also the $n|1|(J6)|\bar{F}$ problem, where we order according to the $(\sum p_k) / n_i$ ratio.

Returning to the more general problem, we find that the only known algorithm is restricted to the case where the directed graph representing the precedence constraints is a forest, i.e. a collection of trees, each with a root node, from one of which runs a path to every other node in the graph.



*) If the order is not given, we first order G_i by previous theorems.

An example is shown above; for the first job there are only two candidates, the jobs 1 and 2. When one of them is scheduled, we delete the node and all the branches leading from it from the tree and get a new set of trees with roots to choose the next job from.

The $n|1|(J6)|\Sigma\alpha_k F_k$ problem in this solution is now solved by Horn's algorithm ([50]). To describe it, we introduce the notion of a successor set S_k to node J_k ; this set has the following properties:

- (1) $J_k \in S_k$;
- (2) if $J_j \in S_k$ and $j \neq k$, then J_k precedes J_j ;
- (3) if $J_j \in S_k$ and J_i precedes J_j , then either $J_i \in S_k$ or J_i also precedes J_k .

Now the algorithm runs as follows. For each root J_k we calculate γ_k . For each root J_k we calculate

$$\gamma_k = \min_{S_k} (\Sigma p_k) / (\Sigma \alpha_k)$$

where the minimum is taken over all successor sets. Schedule the root job with minimal γ_k , remove it and repeat with the new set of roots.

The proof of correctness of this algorithm is extremely complicated. What one does here basically, however, is to find out whether the ordering according to increasing p_k/α_k ratio conflicts with the precedence constraints^{*)}. If this happens, one has to group jobs together, assigning them processing time Σp_k and weight $\Sigma \alpha_k$, in accordance with the result mentioned above.

*) For a more general result on this situation, see Gapp, Mankekar and Mitten [105].

An interesting feature of Horn's method is that it can be also used for situations where the precedence constraints have the form of upside down trees. One just turns the trees upside down again, reverses all arrow and replaces α_k by $-\alpha_k$!

We conclude this section by remarking that no good algorithm is known for the situation in which there are more general precedence constraints among the jobs.

4.2.2. Criteria based on due-dates

We have seen already that the $n|1||\bar{L}$ problem is solved by theorem 4.2.1.A. (order by increasing p_k 's) and that the $n|1||\sum\alpha_k L_k$ problem is solved by theorem 4.2.1.B. (order by increasing p_k/α_k ratio's).

The $n|1||L_{\max}$ and $n|1||T_{\max}$ problems are solved by the following theorem, due to Jackson (reported in [24]):

Theorem 4.2.2.A: the $n|1||L_{\max}$ problem and the $n|1||T_{\max}$ problem are solved by the sequence i_1, \dots, i_n where

$$d_{i_1} \leq d_{i_2} \leq \dots \leq d_{i_n}$$

d_k being the due-date of J_k .

Proof: suppose $d_{i_k} > d_{i_{k+1}}$. Interchanging the two jobs leaves everything unchanged except for the lateness of the k^{th} and $(k+1)^{\text{th}}$ job, the lateness of the $(k+1)^{\text{th}}$ job in the first sequence dominating all the others. The second sequence can therefore not be worse with regards to L_{\max} , nor with regards to $T_{\max} = \max(0, L_{\max})$.

Analogous to theorem 4.2.2.A., one can prove (Conway, Maxwell and Miller [24]):

Theorem 4.2.2.C: the $n|1||L_{\min}$ problem and the $n|1||T_{\min}$ problem are solved by the sequence i_1, \dots, i_n , where

$$d_{i_1} - p_{i_1} \leq d_{i_2} - p_{i_2} \leq \dots \leq d_{i_n} - p_{i_n} .$$

Having solved the $n|1||\sum \alpha_k E_k$ problem in 3.4. by means of dynamic programming and noticing that E_{\max} can be maximized by arguments similar to theorem 4.2.2.B., we can now turn to the more complicated $n|1||\bar{T}$ and $n|1||\sum \alpha_k T_k$ problems.

There are a few situations in which these problems are trivial. If the jobs are all late when scheduled by increasing p_k 's, then in this case the SPT sequence also solves the $n|1||\bar{T}$ problem. Also, if only one job is late when we schedule by increasing due-dates, this sequence solves the $n|1||\bar{T}$ problem in this particular case. However, for a long time these were the only results known.

The first serious work on this problem has been done by Lawler [57], an early article by McNaughton [65] in fact only solving the trivial case that $d_1 = \dots = d_n = 0$.

Lawler has tried out various methods on the more general problem of minimizing $\sum c_k(t)$, where $c_k(t)$ is a monotone non-decreasing cost function. In the first place, he has given a dynamic programming formulation.

If $J \subset N = \{1, \dots, n\}$, define $C(J)$ to be the minimal total cost of performing J , if none of these jobs is started before $d(J) = \sum_{k \notin J} p_k$. Then:

$$\begin{cases} C(J) = \min_J \{c_k(p_k + d(J)) + C(J - \{k\})\} \\ C(\emptyset) = 0 \end{cases}$$

These two equations define a dynamic programming approach, whereby the minimum cost $C(N)$ can be determined. The number of calculations is of the order 2^n and grows therefore very rapidly.

Next, if all the jobs have the same processing time p , Lawler shows that we have a linear programming problem of the assignment type:

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^n \sum_{j=1}^n c_i(jp) x_{ij} \\ & \text{subject to} && \sum_{j=1}^n x_{ij} = 1 \quad (i = 1, \dots, n) \dots\dots\dots (*) \\ & && \sum_{i=1}^n x_{ij} = 1 \quad (j = 1, \dots, n) \dots\dots\dots (**) \\ & && x_{ij} > 0 \quad (i = 1, \dots, n; j = 1, \dots, n) \end{aligned}$$

Here $x_{ij} = 1$ means that job i finishes at time jp .

Finally, Lawler extends this method to the case of different processing times. However, job splitting can not be prevented then. Adding constraints to do so leads to a mixed integer programming formulation.

Lawler and Moore extend their dynamic programming approach, already presented in 3.4., to the $n|1||\sum \alpha_k T_k$ problem where all deadlines are identical.

More interesting, however, is the theoretical work done by Emmons [30] on the $n|1||\bar{T}$ problem. Defining A_k and B_k to be the jobs that have been shown to come after J_k , respectively before J_k in some optimal schedule and ordering the jobs so that $j < k$ implies $p_j \leq p_k$, he proves:

Theorem 4.2.2.C: (i) if $j < k$ and $d_j \leq \max(\sum_{i \in B_k} p_i + p_k, d_k)$, then J_j comes before k in some optimal schedule;

(ii) if $d_1 \leq \max(p_k, d_k)$ for all $k > 1$, then J_1 comes first in an optimal schedule;

(iii) if $\max(p_n, d_n) \geq d_k$ for all $k < n$, then J_n is last in an optimal schedule;

(iv) if SPT scheduling is identical with earliest due-date scheduling, then these schedules are optimal;

(v) the SPT schedule is optimal if

$$d_k + p_k \leq \sum_{i=1}^{k+1} p_k \text{ for } k = 1, \dots, n-1.$$

Proof: (i) \Rightarrow (ii), (i) \Rightarrow (iii): take $B_k = \emptyset$

(i), (ii) \Rightarrow (iv): trivial

(ii) \Rightarrow (v): $d_1 < p_2$, so J_1 is first. Removing it and subtracting p_1 from all d_k , J_2 must be first in the new job set, etc..

So we only have to prove (i), which is possible by carefully considering the effect of interchanging J_k and J_j (see Emmons [30]).

The next theorem tells when a longer job may precede a shorter one.

Theorem 4.2.2.D: (i) if $j < k$, $d_j > \max(\sum_{i \in B_k} p_i + p_k, d_k)$ and $d_j + p_j > \sum_{i \in A_k} p_i$, then k precedes j in some optimal sequence;

(ii) if $d_k = \max d_j$ and $d_k + p_k > \sum_{t=1}^n p_i$, then J_k is last in an optimal schedule;

(iii) the earliest due-date schedule is optimal if $L_k < p_k$ for all k .

Proof: (i) \Rightarrow (ii), together with theorem 4.2.2.C.(i).

(ii) \Rightarrow (iii): if $d_k = \max d_j$, and J_k is last, then:

$L_k = \sum_{i=1}^n p_i - d_k < p_k$ implies we can use (ii), drop J_k and repeat.

(i) is proved again by looking at the effect of putting J_k directly in front of J_j .

Emmons gives a branch-and-bound algorithm based on reducing the search for an optimum as much as possible by means of the two theorems above and branching when it cannot be determined if one job proceeds another or not. He gives no details, no computer results and no bounding prescription. However, we shall illustrate the use of his theorems by a small example.

Suppose:

$$\begin{cases} p_1 = 1 & p_2 = 3 & p_3 = 4 & p_4 = 9 & p_5 = 15 \\ d_1 = 11 & d_2 = 6 & d_3 = 14 & d_4 = 10 & d_5 = 9 \end{cases}$$

Now we find:

(1) J_5 is last, because $\max(p_5, d_5) = 18 \geq d_k$ for $k = 1, 2, 3, 4$ (theorem 4.2.2.C.(iii)). We remove J_5 , getting:

$$\begin{array}{cccc} p'_1 = 1 & p'_2 = 3 & p'_3 = 4 & p'_4 = 9 \\ d'_1 = 11 & d'_2 = 6 & d'_3 = 14 & d'_4 = 10 \end{array}$$

(2) Theorem 4.2.2.C.(iii) cannot be applied again: $\max(9, 10) = 10 < 11, 17$. However, $\max\{d'_j\} = d'_3 = 14 > p'_1 + p'_2 + p'_4 = 13$, so now we put J_3 last because of theorem 4.2.2.D.(ii). We get:

$$\begin{array}{ccc} p''_1 = 1 & p''_2 = 3 & p''_3 = 9 \\ d''_1 = 11 & d''_2 = 6 & d''_3 = 10 \end{array}$$

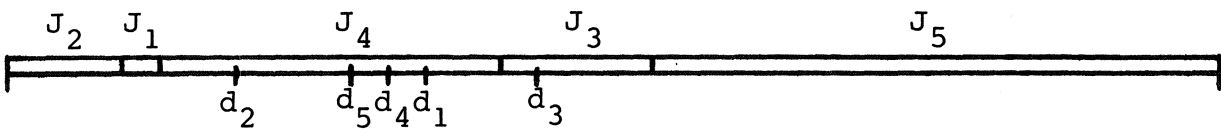
(3) We cannot reapply theorem 4.2.2.C.(iii) ($10 < 11$), nor can we reapply theorem 4.2.2.D.(ii) ($11 < 12$). Now look at J_1 . If $d''_1 < \max(d''_2, p''_2)$ or $d''_1 < \max(d''_3, p''_3)$, then J_1 would precede J_2 or J_3 by theorem 4.2.2.C.(i). However, this is not the case. We see next that J_2 precedes J_3 : $6 < \max(9, 10)$. So $J_3 \in A_2$, and J_1 and J_2 are candidates for the first place.

(4) Now $d_1'' > \max(p_2'', d_2'')$ and $d_1'' + p_1'' \geq p_1'' + p_2''$. So, by theorem 4.2.2.D.(i), J_2 precedes J_1 . We remove J_2 , putting it first and subtract $p_2'' = 3$ from d_1'' and d_3'' ; we get:

$$\begin{aligned} p_1''' &= 1 & p_2''' &= 9 \\ d_1''' &= 8 & d_2''' &= 7 \end{aligned}$$

(5) By theorem 4.2.2.C.(i), J_1 precedes J_2 , because $d_1''' = 8 \leq 9 = \max(p_2''', d_2''')$.

So the optimal order is: $J_2 - J_1 - J_4 - J_3 - J_5$,



with average tardiness $T = \frac{1}{5} (0 + 0 + 3 + 3 + 23) = 29/5$.

In this case, branching has not been necessary. It is difficult to judge Emmons' algorithm, because the branch-and-bound details are so insufficiently specified.

Finally and most recently, a branch-and-bound solution to the $n|1||\sum \alpha_k T_k$ problem has been suggested by Shwimer [91], inspired by work of Elmaghraby.

This branch-and-bound algorithm constructs an optimal sequence in the inverse order. The first subsets are formed by taking successively J_1, \dots, J_n as the last job, ordering the other jobs by increasing due-dates and keeping the best schedule. However, here as during the whole algorithm the following elimination theorem is used:

Theorem 4.2.2.E: if $\alpha_j \geq \alpha_k$, $d_j \leq d_k$ and $p_j \leq p_k$, then one only has to consider schedules where J_j precedes J_k .

Proof: this is a direct extension of theorem 4.2.2.C.(i) and is proved in a likewise complicated check of all possibilities when J_k and J_j are interchanged.

At any level, we branch by means of the set S of jobs not yet scheduled. If S consists of only one job, we can construct a complete solution; if $d_k = \max_S d_j \geq \sum_{j \in S} p_j$, then place J_k last of all jobs in S and branch from $S - \{J_k\}^*$; otherwise, create subsets by successively placing each job J_k last among all jobs in S .

A lower bound LB is then given by the following expression:

$$LB = C + \alpha_k \max_S ((\sum p_i) - d_k, 0) +$$

$$\min_{S-\{J_k\}} \{ \alpha_j \max_S ((\sum p_i) - p_k - d_j, 0) +$$

$$(\min_{S-\{J_k, J_j\}} \alpha_i) \cdot T_{\max}(S - \{J_k, J_j\}) \}$$

where C = cost incurred so far (see below), and the whole lower bound is based on the idea of scheduling just before J_k that job J_j which adds the smallest possible amount to $\sum \alpha_k T_k$, and then finding the minimal T_{\max} of the remaining jobs by scheduling them according to due-date (theorem 4.2.2.B.), multiplying this by the smallest remaining weight α_i .

The costs C incurred so far are stored with any subset and calculated by adding $\alpha_k \max_S ((\sum p_i) - d_k, 0)$ to the previously incurred cost. The algorithm is of the "newest active node" type, the subsets being stored away, however, in order of decreasing lower bounds. Computer experience is quite good, a $30|1||\sum \alpha_k T_k$ problem being solved in about 4 seconds on an IBM 360/65. Still, sharper bounds and more extensive theoretical elimination may well speed up things considerably.

Generally we may conclude that the non-linearity of T_k causes serious theoretical complications in the $n|1||\bar{T}$ and $n|1||\sum \alpha_k T_k$

*) This uses in fact a weaker form of theorem 4.2.2.D.(ii).

problems, and that the situation with regards to solutions is still far from satisfactory.

4.2.2.1. Precedence constraints

Suppose now there are precedence constraints among the jobs. The $n|1|(J6)|T_{\max}$ problem can be dealt with by a theorem of Lawler and Moore [59]. However, Lawler has since then given a quicker and more general method for these and other problems [60]. Suppose a monotone non-decreasing function $c_k(t)$ is given, describing the loss incurred if J_k finishes at time t .

Theorem 4.2.2.B: let S be the subset of jobs not required to precede any others, and $T = \sum_{k=1}^n p_k$. If K is such that

$$c_K(T) = \min_{k \in S} \{c_k(T)\},$$

then there exists a sequence minimizing the maximum loss, where J_K is last.

Proof: if J_K is not last, putting it last can never increase the maximum loss.

This theorem solves our problem: for $n|1|(J6)|L_{\max}$, take $c_k(t) = t - d_k$; for $n|1|(J6)|T_{\max}$, take $c_k(t) = \max(t - d_k, 0)$.

In view of the complicatedness of the $n|1||\bar{T}$ and $n|1||\sum \alpha_k T_k$ problems, it is not surprising that no work has been done on these problems if there are precedence constraints among the jobs as well.

4.2.2.2. Number of tardy jobs

We end this section by considering a slightly different problem: minimizing the number of tardy jobs. Moore [69] gave an algorithm to solve this problem, which was simplified by Hodgson to read as follows:

- (1) sequence the jobs according to increasing due-dates, giving a sequence i_1, \dots, i_n ;
- (2) if all jobs are on time, we have finished;
- (3) if J_{i_k} is the first late job, remove J_{i_ℓ} , where $p_{i_\ell} = \max(p_{i_1}, \dots, p_{i_{\ell-1}})$, to be processed later. Repeat (2) and (3) on the remaining sequence, until no remaining jobs are late anymore.

Moore's proof of the correctness of the algorithm, was fairly difficult and has been simplified later by Sturm [97].

We can also apply the functional equation of 3.4. here, ordering the jobs according to due-date and specifying:

$$c_k(t) = \begin{cases} 0 & t \leq d_j \\ 1^*) & t > d_j \end{cases}$$

$$g_k = p_k \qquad \gamma_k(t) = 0$$

$$s_k = 0 \qquad s_k(t) = 1^*)$$

*) Or α_k , if the weighted number of tardy jobs is to be minimized.

4.2.3. Criteria based on machine utilization

For the sake of completeness, we just remark that, in the $n|1$ case, F_{\max} being a constant, the mean number of jobs in the shop is directly proportional to \bar{F} , and is therefore minimized by SPT scheduling.

4.2.4. Criteria based on change-over times

As announced already in 2.3.4., we shall treat two cases of interest here. The most well known one is surely the $n|1$ problem, whereby it takes c_{ij} time units if job j is followed by job i . Minimizing total change-over time is then equivalent to finding the sequence i_1, \dots, i_n that minimizes

$$\sum_{k=1}^{n-1} c_{i_k i_{k+1}}$$

This problem will be readily recognized as the famous Travelling Salesman Problem, where a salesman has to visit n cities with distances c_{ij} and wants to minimize the distance he travels. Bellmore and Nemhauser [14], amongst others, give a survey of known solution methods; no generally efficient algorithm is known, but certain branch-and-bound algorithms can solve problems up to 80 cities. A special case, originating as a

machine scheduling problem, where $c_{ij} = \int_{B_i}^{A_j} f(x)dx$ if $A_j \geq B_i$ and $c_{ij} = \int_{A_j}^{B_i} g(x)dx$ if $B_i > A_j$, with (A_k, B_k) given constants and $f(x) + g(x) \geq 0$ for all x , has been solved by Gilmore and Gomory [40].

In the second place we want to draw attention to a problem, that strictly speaking does not fall within our definition of the machine scheduling problem. The problem, considered

by Glassey [41], consists of finding the minimum number of change-overs needed to have produced $d_k(t)$ units of product k by time t ($t = 1, \dots, T$) where the machine produces one unit of product per time unit. The problem not quite belonging in our class, we shall not pay much attention to it here, but only formulate the elegant result, obtained by graph-theoretical and dynamic programming arguments. Denoting $(d_1(T), \dots, d_n(T))$ as p_0^* , we generally construct the set p_i^* by including all elements (x_1^i, \dots, x_n^i) where for some $(x_1^{i-1}, \dots, x_n^{i-1}) \in p_{i-1}^*$ and for some $k : x_1^i = x_1^{i-1}, \dots, x_{k-1}^i = x_{k-1}^{i-1}, x_{k+1}^i = x_{k+1}^{i-1}, \dots, x_n^i = x_n^{i-1}$ and $x_k^i = x_k^{i-1} - y$, where y is maximal in the sense that $x_k^{i-1} - y - 1$ would be smaller than:

$$\begin{aligned} & d_k[T - (d_1(T) - x_1^{i-1}) - \dots - (d_{k-1}(T) - x_{k-1}^{i-1}) - \\ & - (d_k(T) - x_k^{i-1} - y - 1) - (d_{k+1}(T) - x_{k+1}^{i-1}) - \dots - \\ & - (d_n(T) - x_n^{i-1})] \end{aligned}$$

(the level of demand at that time).

In this sense (x_1^i, \dots, x_n^i) is one of the best predecessors of $(x_1^{i-1}, \dots, x_n^{i-1})$.

Proceeding like this and eliminating obviously non-optimal points in p_i^* , the optimum value is the minimal i for which $(0, 0, \dots, 0) \in p_i^*$, as can be easily understood.

4.2.5. Multiple criteria

We shall not delve into general methods for combining various criteria (see Ashour [2] for examples), but shall only treat two problems, where some criterium is minimized, subject to the condition that no job may be finished after its due-date.

The oldest of the two, solved by Smith [93], is the $n|1||\Sigma\alpha_k F_k, T_{\max} = 0$ problem.

Theorem 4.2.5.A: if there is a sequence whereby $T_{\max} = 0$ *) , then J_K is in last position in the solution to the $n|1||\Sigma\alpha_k F_k, T_{\max} = 0$ problem if:

$$(1) \quad d_K \geq \sum_{k=1}^n p_k \quad (\text{i.e., } T_K = 0);$$

$$(2) \quad p_K / \alpha_K \geq p_L / \alpha_L \quad \text{for all } L \text{ with } d_L \geq \sum_{k=1}^n p_k$$

(i.e., J_K has the greatest p_K / α_K ratio of all jobs that could be last).

Proof: if $d_L < \sum_{k=1}^n p_k$, then putting J_L last will make $T_L > 0$.

If $d_L \leq \sum_{k=1}^n p_k$, then putting it last would increase $\Sigma\alpha_k F_k$

(theorem 4.2.1.B.). (Having put J_K last, we repeat the procedure for the other $(n-1)$ jobs).

Bratley, Florian and Robillard [17] solve two problems: the $n|1|(16)|F_{\max}, T_{\max} = 0$ and $n|1||F_{\max}, T_{\max} = 0$.

The first one, where job splitting is allowed, is fairly simple. Let x_{ij} be a 0 - 1 variable, $x_{ij} = 1$ indicating that job j is processed at time i . Then we have:

$$\sum_{j=1}^n x_{ij} \leq 1 \quad (i = 1, \dots, \max_k \{d_k\}) \quad (\text{at most one job at a time})$$

$$\sum_{i=r_j}^{d_j} x_{ij} = p_j \quad (j = 1, \dots, n)$$

*) If sequencing according to increasing d_k does not accomplish this, no sequence will (theorem 4.2.2.B.)!

$$\begin{aligned}
 x_{1j} &= x_{2j} = \dots = x_{r_j j} = x_{d_j, j} = x_{d_{j+1}, j} = \dots = \\
 &= x_{\max \{d_j\}, j} = 0 \quad (j = 1, \dots, n).
 \end{aligned}$$

A feasible solution can be found by a labelling algorithm similar to that for the assignment problem. By checking if $\sum_{j=1}^n x_{ij} = 1$ for all i , one can find out if there is any point in looking for a better solution.

The second problem, where job splitting is not allowed, is solved by a branch-and-bound algorithm.

We create initial subsets by putting either J_1, J_2, \dots or J_n first in the sequence. No direct bound is calculated; we branch from every subset, and split them up by placing successively all jobs that are still unscheduled, in the next position. There are, however, several exclusion mechanisms:

Lemma 4.2.5.A.: if we schedule a job in a particular position and it is then late, we may disregard the entire subset.

Lemma 4.2.5.B.: if we schedule J_j , where r_j is larger than the sum of processing times of all scheduled jobs, the sequence is so far optimal and we need not consider reordering the jobs scheduled so far.

Lemma 4.2.5.C.: if we have a feasible solution where: (i) some jobs J_K starts at r_K ; (ii) all the following jobs are processed without delay; (iii) all their r_j 's are larger than r_K , then this solution is optimal.

Lemma 4.2.5.D.: if a feasible solution with $F_{\max} = t$ is not optimal, we may set all d_j 's equal to $(t-1)$.

Computer experience with this algorithm is good: 66% of a set of 100 job problems is solved within 18 seconds ($100! \approx 10^{158}$) by a CDC 6400 computer.

4.2.6. Multiple identical parallel machines

To end this section on the one-machine problem, we pay attention to the situation logically belonging here, where a job can be processed on any of m identical machines. Simultaneous processing is impossible. The machines are identical in the sense that p_k is independent of the machine on which J_k is processed.

Here, preemption may very well speed up things considerably, so every time we have to consider whether it will be allowed or not.

It is easy to see that

$$F_{\max} = \max \left(\max_k \{p_k\}, \frac{1}{m} \sum p_k \right)$$

(McNaughton [65]), if preemption is allowed. If not, the optimal schedule may be difficult to find; no satisfactory solution exist for this problem. Baker ([5]) reports a heuristic procedure that behaves fairly well.

Fortunately, for the \bar{F} and $\sum \alpha_k F_k$ criteria, McNaughton [65] has proved that there exists an optimal schedule without splits.

The $n|m_p||\bar{F}$ problem is now trivially solved: if one machine ℓ the job sequence is $J_{\ell_1}, \dots, J_{\ell_{n_\ell}}$, then

$$\bar{F} = \frac{1}{n} \sum_{\ell=1}^m \sum_{k=1}^{n_\ell} (n_\ell - k + 1) p_{\ell_k}.$$

Applying the results used in theorem 4.2.1.A., we see that we can order all jobs by increasing p_k and schedule the first m jobs first on M_1, \dots, M_m , the next m jobs second on M_1, \dots, M_m , etc..

No such satisfactory solution exists for the $n | m_p | | \sum \alpha_k F_k$ *) problem, however. Clearly, on each machine jobs have to be ordered by increasing p_k/α_k ratio, but it is not at all obvious how to divide the jobs over the machines. Apart from two heuristic methods reported by Baker ([5]) and a generalisation of Lawler's LP method (reported in 4.2.2.), leading to a transportation problem if all processing times are equal, the best we can do is to use dynamic programming (Rothkopf [85]). First, we order the jobs according to decreasing p_k/α_k ratio's, so that it is no restriction to assume that all machines will process any subset of them in the inverse order. We then define $C_k(t_1, \dots, t_m)$ to be the minimal cost of processing J_1, \dots, J_k if m_ℓ is not available before t_ℓ . Then:

$$C_k(t_1, \dots, t_m) = \min_{\ell} \{ C_{k-1}(t_1, \dots, t_\ell + p_k, \dots, t_m) + \alpha_k(t_\ell + p_k) \}$$

where we start by evaluating $C_1(t_1, \dots, t_m)$ for all combinations (t_1, \dots, t_m) for which $t_\ell < t_{\ell+1}$ ($\ell = 1, \dots, m-1$) and

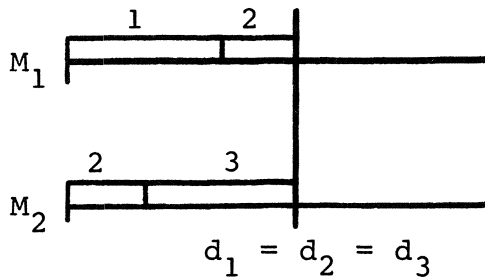
$$\sum_{\ell=1}^n t_\ell = \sum_{k=2}^n p_k, \text{ and take } C_0(t_1, \dots, t_m) = 0^{**}.$$

No computing results are given, but the procedure is most likely very time-consuming.

Where the tardiness criteria cause already so many difficulties in the one-machine case, it is hardly surprising that for m parallel machines there are very few results indeed. Job splitting may very well be profitable here, as demonstrated by the \bar{T} optimal schedule below.

*) M_p is to be read: m parallel machines.

**) Rothkopf treats a slightly more general case than we do.



Again we can use Lawler's transportation problem-formulation if all p_k are equal. If this is not the case, we are left with Root's algorithm [84] that solves the $n|m_p||\bar{T}$ problem where all deadlines are equal and job-splitting is not allowed. Root attacks this problem by theoretical arguments. His proofs are again very involved; basically his method boils down to the following. Ordering the jobs first by increasing processing-times, Root proves that there is an optimal sequence whereby J_1, \dots, J_q are all started before the common due-date d , and J_{q+1}, \dots, J_n are processed by scheduling $J_{q+1}, \dots, J_{q+1+m}$ next on M_1, \dots, M_m , followed by $J_{q+1+m+1}, \dots, J_{q+1+2m}$ on M_1, \dots, M_m , until every job has been scheduled. The only problem then is to determine q , and the schedule J_1, \dots, J_q . As to q , the only result given is that the number of feasible values for q is smaller than m . For every feasible value of q , we have to find the schedule for J_1, \dots, J_q that minimizes $\sum_{\ell=1}^m T_\ell$, where T_ℓ is the tardiness of the (maximally one) job that finishes late on M_ℓ . No algorithm is given for this procedure either. The lack of these details make it very difficult to judge the computational value of Root's work.

We conclude that algorithms for the $n|m_p||\sum \alpha_k F_k$ and the $n|m_p||\sum \alpha_k T_k$ problems are very much needed and are likely to be fairly complicated.

4.2.6.1. Precedence constraints

The only problem with precedence constraints that has been satisfactorily attacked, is the $n|m_p|(J6)|F_{\max}$ one, where job splitting is not allowed and $p_k = 1$ for all k . We then have Hu's algorithm [51]: label all jobs without successors 1; then label the other jobs α_k where

$$\alpha_k = 1 + \max \{ \alpha_j | J_k \text{ directly-precedes } J_j \text{ by the precedence constraints} \}.$$

Now, if there are less than m scheduleable jobs, schedule them; otherwise schedule the m jobs with largest α_k . Repeat until all jobs are scheduled. Again we find here that the correctness of this intuitively obvious algorithm is very hard to prove.

Treating J_k with processing time p_k as a series of p_k jobs with processing time 1, we can apply this algorithm to the more general case too. A limited kind of preemption can then not be avoided. If we allow preemption in general, only the case where $m = 2$ has been solved by Muntz and Coffman [23] (reported by Baker [5]). Their algorithm boils down to splitting J into independent (non-interfering) subsets, ordering these first and then combining them; it is fairly complicated and we shall not repeat it here, as no generalisation for larger m seems possible anyhow. The general problem, either with or without preemption, remains unsolved. Also no other criteria have been investigated here.

With regards to the $n|m_p$ problems therefore, a lot of work remains to be done. The difficulty here consists chiefly of assigning the jobs to a machine; at present, no generally satisfactory rule for this procedure exists.

4.3. The two-machines problem ($n|2$)

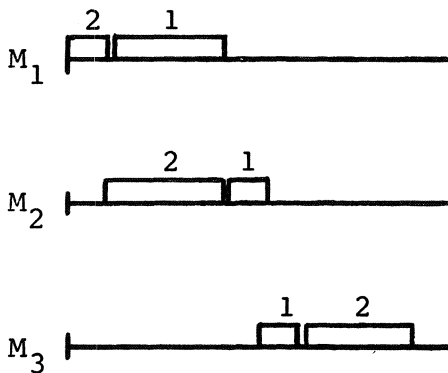
We start this section by proving two theorems (Conway, Maxwell and Miller [24]) that will drastically reduce the number of potentially optimal solutions to some future problems.

Theorem 4.3.A: in solving $n|m|F|\phi(c_1, \dots, c_n)$ problems, where ϕ is any regular measure, we only have to consider schedules on which the same job order is prescribed on the first two machines.

Proof: trivial.

Theorem 4.3.B: in solving $n|m|F|F_{\max}$ problems, we only have to consider schedules with the same job order on M_1 and M_2 , and the same job order on M_{m-1} and M_m .

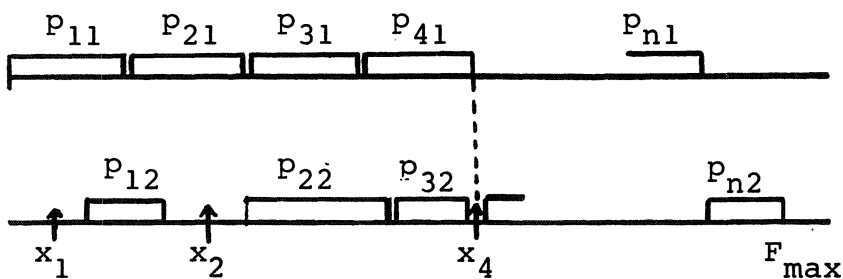
Proof: interchanging jobs on M_m will not increase F_{\max} . (Theorem 4.3.B.) is not true for any regular measure; look at the \bar{F} -optimal schedule below!).



Several applications of these theorems will be given below.

4.3.1. The $n|2|F|F_{\max}$ -problem

This is the problem solved in the often-mentioned paper by Johnson []. Because of theorem 4.3.A., we may restrict ourselves to schedules prescribing the same order on M_1 and M_2 . A Gantt chart could look as follows:



Theorem 4.3.1.A: the $n|2|F|F_{\max}$ problem is solved by the sequence i_1, \dots, i_n , where

$$\min (p_{i_k 1}, p_{i_{k+1} 2}) < \min (p_{i_{k+1} 1}, p_{i_k 2}).$$

Proof: it is trivial that we must minimize $\sum_{j=1}^n X_{i_j}$ over all sequences (i_1, \dots, i_n) (where X_{i_j} is idle time on M_2 between the processing of $J_{i_{j-1}}$ and J_{i_j}), and that

$$\begin{aligned} \sum_{j=1}^n X_{i_j} &= \max \left(\sum_{j=1}^n p_{i_j 1} - \sum_{j=1}^{n-1} p_{i_j 2}, \sum_{j=1}^{n-1} p_{i_j 1} - \sum_{j=1}^{n-2} p_{i_j 2}, \dots, p_{i_1} \right) \\ &= \max (K_n, \dots, K_1) \end{aligned} \quad (*)$$

$$\text{where } K_k \stackrel{\text{def}}{=} \sum_{j=1}^k p_{i_j 1} - \sum_{j=1}^{k-1} p_{i_j 2}.$$

Now, interchanging i_k and i_{k+1} , changes K_k into K'_k , K_{k+1} into K'_{k+1} and leaves the other K_k 's unaffected.

The old sequence will be better if

$$\max (K_k, K_{k+1}) < \max (K'_k, K'_{k+1}) \quad (**)$$

Now, $\max (K_k, K_{k+1})$ is given by (*), and:

$$\begin{aligned} \max (K'_k, K'_{k+1}) &= \\ \max \left(\sum_{j=1}^{k-1} p_{i_j 1} + p_{i_{k+1} 1} - \sum_{j=1}^{k-1} p_{i_j 2}, \right. \\ &\quad \left. \sum_{j=1}^{k+1} p_{i_j 1} - \sum_{j=1}^{k-1} p_{i_j 2} - p_{i_{k+1} 2} \right) = \\ \sum_{j=1}^{k+1} p_{i_j 1} - \sum_{j=1}^{k-1} p_{i_j 2} + \max (-p_{i_k 1}, -p_{i_{k+1} 2}). \end{aligned}$$

So, treating $\max (K_k, K_{k+1})$ likewise, (**) will be true if:

$$\max (-p_{i_{k+1}1}, -p_{i_k2}) < \max (-p_{i_k1}, -p_{i_{k+1}2})$$

leading easily to:

$$\min (p_{i_k1}, p_{i_{k+1}2}) < \min (p_{i_{k+1}1}, p_{i_k2})^{**}$$

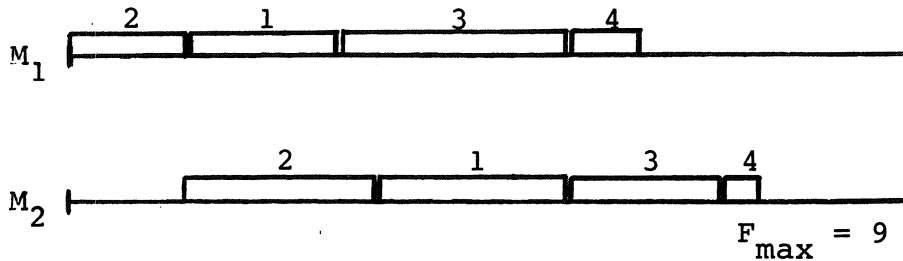
An example shows that theorem 4.3.1.A. is intuitively plausible.

$$\begin{array}{cccc} p_{11} = 4 & p_{21} = 3 & p_{31} = 6 & p_{41} = 2 \\ p_{12} = 5 & p_{22} = 5 & p_{32} = 4 & p_{42} = 1 \end{array}$$

The steps are:

- (1) p_{42} is minimal, so J_4 comes last; cross off J_4^{**} ;
- (2) now p_{21} is minimal, J_2 comes first and can be removed;
- (3) p_{11} is minimal now; J_1 comes before J_3 .

Solution:



*) Of course, we do apply theorem 3.6.A. here; formally we would have to prove that this function g defines a transitive relation.

***) Ties between p_{k_1} and p_{l_2} are resolved in favour of J_k .

A graphical interpretation of the algorithm is given in Conway, Maxwell and Miller [24].

4.3.2. The $n|2|F|\bar{F}$ problem

No comparably easy solution is known for the $n|2|F|\bar{F}$ problem, although theorem 4.3.A. still is applicable. It is easy to see that J_{i_k} should precede J_{i_ℓ} if both $p_{i_k 1} \leq p_{i_\ell 1}$ and $p_{i_k 2} \leq p_{i_\ell 2}$. However, this does not order the jobs. Ignall and Schrage [52] offer a "newest-active-node" branch-and-bound algorithm. Branching is done by next scheduling any job that is not yet scheduled. The bound is given by adding the flow times of jobs already completed to $\max(S, T)$, where S and T are the sum of remaining flow times under the respective assumptions that $p_{i_k 1} \geq p_{i_k 2}$ and $p_{i_k 2} \geq p_{i_k 1}$ for all unscheduled jobs. The computer results are quite discouraging; if $n = 9$, a difficult problem took 4 minutes on a CDC 1604. The number of computations grows exponentially with n . However, no better algorithms are known.

4.3.3. The $n|2|G|F_{\max}$ problem

The $n|2|G|F_{\max}$ problem was solved in an ingenious way by Jackson [53], and surprisingly enough, approximately seven years later in a less ingenious way by Szwarc [99]. Jackson's solution is simply to divide all the jobs in four groups:

G_i contains the jobs that are only processed on M_i
($i = 1, 2$);

G_{ij} contains the jobs that are processed first on M_i ,
then on M_j ($i = 1, j = 2$; $i = 2, j = 1$).

Now, sequence the jobs in G_{12} and G_{21} according to Johnson's method, and choose the following order:

on M_1 : jobs from G_{12} followed by
 jobs from G_1 followed by
 jobs from G_{21} ;
 on M_2 : jobs from G_{21} followed by
 jobs from G_2 followed by
 jobs from G_{12} .

This order is clearly optimal.

4.3.4. Miscellaneous two-machine problems

Mitten [68], Johnson [55], Szwarc [99] and Nabeshima [74] have considered $n|2|F|F_{\max}$ problems wherein some time lags between operations on M_1 and M_2 have been prescribed.

In Mitten's paper, constants l_k ($k = 1, \dots, n$) are given. The operation (J_k, M_2) may start l_k time-units after (J_k, M_1) has started (if M_2 is free then); however, it must not be finished sooner than l_k time-units after the finishing time of (J_k, M_1) . Overlapping is therefore allowed.

Denoting starting-times of (J_k, M_ℓ) ($k = 1, \dots, n$; $\ell = 1, 2$) by $t_{k\ell}$, and restricting ourselves to "passing not permitted", we see:

$$t_{k1} = t_{k-1,1} + p_{k-1,1}$$

$$t_{k2} = \max (t_{k-1,2} + p_{k-1,2}, t_{k1} + l_k, t_{k1} + p_{k1} + l_k - p_{k2}).$$

Just as in 4.3.1., we have to minimize $\sum_{k=1}^n X_{i_k}$ the total idle time on M_{23} over all sequences i_1, \dots, i_n .

Now define: $y_k = l_k - \min (p_{k1}, p_{k2})$. Following Johnson [55], we may interpret y_k as the (possibly negative and possibly

overlapping) processing times on J_k by an intermediate machine. So the $n|2|P, (20)|F_{\max}$ problem is equivalent to a $n|3|F|F_{\max}$ problem, which, because of theorems 4.3.A. and 4.3.B. is again equivalent to a $n|3|P|F_{\max}$ problem. However, it is a very special $n|3|P|F_{\max}$ problem. In a general $n|3|F|F_{\max}$ problem we would find for a given sequence i_1, \dots, i_n :

$$F_{\max} = \max_{1 \leq u \leq v \leq n} \left(\sum_{k=1}^u p_{i_k 1} + \sum_{k=u}^v y_k + \sum_{k=v}^n p_{i_k 2} \right)^*.$$

In this case, however, the y_k may overlap (there are "no bottlenecks") and so we have here:

$$F_{\max} = \max_{1 \leq u \leq n} \left\{ \sum_{k=1}^u p_{i_k 1} + y_u + \sum_{k=u}^n p_{i_k 2} \right\}$$

which implies that we can treat this a special case of the $n|2|F|F_{\max}$ case solved by Johnson with processing times:

$$p'_{k1} = p_{k1} + y_k$$

$$p'_{k2} = p_{k2} + y_k$$

(for details on this, see 4.3.).

Sequencing J_1, \dots, J_n according to Johnson's method, using the processing times above, leads to the optimal sequence. By interpreting the problem in this way, we have avoided Mitten's long and complicated proof, that leads, of course, to the same algorithm.

Szwarc [99] and Nabeshima [74] consider slightly other forms of this problem. Szwarc only introduces start-lags $m_k - p_{k1}$ so that, in the notation used above:

$$t_{k1} = t_{k-1,1} + p_{k-1,1}$$

*) See 4.4.

$$t_{k2} = \max (t_{k-1,2} + p_{k-1,2}, t_{k1} + p_{k1}, t_{k1} + m_k).$$

Szwarc proves that the optimal order is identical on both machines and can be found by splitting the jobs in two subsets S and S' where:

$$S = \{J_k \mid z_k = p_{k1} - p_{k2} < 0\}$$

$$S' = \{J_k \mid z_k > 0\}$$

Jobs in S precede jobs in S' . S itself is ordered by increasing values of $\max (p_{k1}, m_k)$ and S' is ordered by decreasing values of $\max (p_{k2}, m_k - z_k)$. Szwarc's proof is very involved. However, we may again regard $\max (m_k - p_{k1}, 0)$ as the processing time of J_k on a non-bottleneck intermediate machine. Therefore, we can apply Johnson's algorithm again, putting:

$$p'_{k1} = p_{k1} + \max (m_k - p_{k1}, 0)$$

$$p'_{k2} = p_{k2} + \max (m_k - p_{k1}, 0).$$

Nabeshima deals with the situation that (J_k, M_1) is split in two consecutive parts with processing times p^1_{k1} and p^2_{k1} . Moreover, (J_1, M_2) may not be started before $t_{k1} + l_k$, and may not be finished before $t_{k1} + p^1_{k1} + m_k$.

So we have here

$$t_{k1} = t_{k-1,1} + p^1_{k-1,1} + p^2_{k-1,2}$$

$$t_{k2} = \max (t_{k-1,2} + p_{k-1,2}, t_{k1} + l_k, t_{k1} + p^1_{k1} + m_k - p_{k2})$$

Again avoiding Nabeshima's complicated proof, we note that this is equivalent to a special $n|3|F|F_{\max}$ problem, where y_k (the

processing time on the non-bottleneck intermediate machine) is given by:

$$y_k = \max (\ell_k - p_{k1}, m'_k - p_{k2})$$

where $p_{k1} = p_{k1}^1 + p_{k1}^2$ and $m'_k = m_k - p_{k1}^2$

If we drop the assumption that the two machines process the jobs in the same order, things get much more complicated. Johnson [55] gives a method by which one can reduce the set of feasible solutions to potentially optimal ones only. However, a good algorithm remains to be found.

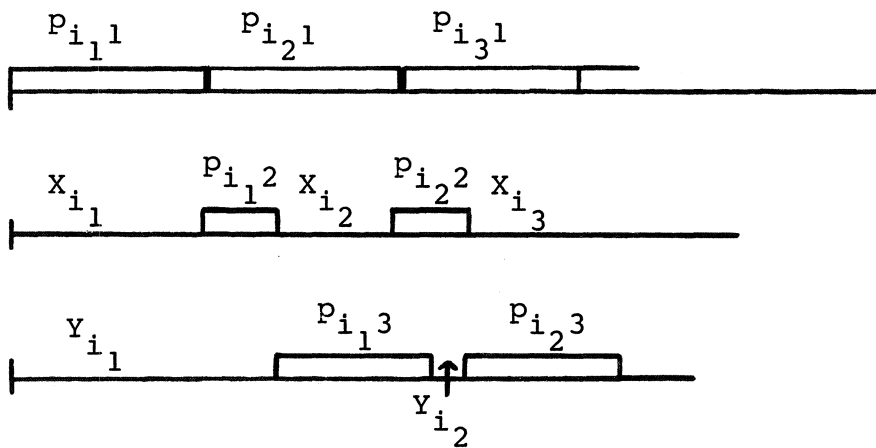
We finish this section of miscellaneous $n|2$ problems by mentioning the work of Sahney [88] on a $n|2|(M3)|\bar{F}$ problem, where jobs J_1, \dots, J_K have to be processed by M_1 only, J_{K+1}, \dots, J_n have to be processed by M_2 , and there is a time ζ_{ij} needed to move the one available machine operator from machine i to machine j ($i, j = 1, 2; i \neq j$). A few theorems are obvious then: ordering the jobs by increasing p_k 's on each machine, we can at any point where J_1, \dots, J_{i-1} and $J_{K+1}, \dots, J_{K+1+(j-1)}$ have been processed so far, stick to M_1 if $p_i < p_{K+1+j}$ and switch to M_2 if $p_i > p_{K+1+j} + \zeta_{12} + \zeta_{21}$. Sahney derives a few more complicated theorems and suggests a branch-and-bound procedure to choose between the remaining feasible solutions.

We shall not go into details here any further, but wish to point out that Sahney's work is one of the few theoretical approaches that explicitly considers labour as a limiting factor.

4.4. The three-machines problem (n|3)

Theoretical results for the case that $m = 3$ center around the $n|3|F|F_{\max}$ problem. Here again we can apply theorem 4.3.B. and conclude that the job order on each machine will be identical in an optimal sequence.

A Gantt chart of any sequence i_1, \dots, i_n will look as follows (the meaning of X_{i_k} and Y_{i_k} here are obvious).



Now, we want to minimize $\sum_{j=1}^n Y_{i_j}$ over all sequences i_1, \dots, i_n .

We have:

$$Y_{i_n} = \max \left(\sum_{j=1}^n p_{i_j 2} + \sum_{j=1}^n X_{i_j} - \sum_{j=1}^{n-1} p_{i_j 3} - \sum_{j=1}^{n-1} Y_{i_j}, 0 \right)$$

so:

$$\begin{aligned} \sum_{j=1}^n Y_{i_j} &= \max \left(\sum_{j=1}^n p_{i_j 2} + \sum_{j=1}^n X_{i_j} - \sum_{j=1}^{n-1} p_{i_j 3}, \sum_{j=1}^{n-1} Y_{i_j} \right) \\ &= \max \left(\sum_{j=1}^n p_{i_j 2} + \sum_{j=1}^n X_{i_j} - \sum_{j=1}^{n-1} p_{i_j 3}, \sum_{j=1}^{n-1} p_{i_j 2} + \sum_{j=1}^{n-1} X_{i_j} \right. \\ &\quad \left. - \sum_{j=1}^{n-2} p_{i_j 3}, \dots, p_{i_1} + X_{i_1} \right) \\ &= \max_{1 \leq u \leq v \leq n} \{H_v + K_u\} \end{aligned}$$

where $H_v = \sum_{j=1}^v p_{i_j 2} - \sum_{j=1}^{v-1} p_{i_j 3}$, $K_u = \sum_{j=1}^u p_{i_j 1} - \sum_{j=1}^{u-1} p_{i_j 2}$

Adding $\sum_{j=1}^n p_{i_j 3}$ to both sides, we find:

$$F_{\max} = \sum_{j=1}^n p_{i_j 3} + \sum_{j=1}^n Y_{i_j} = \max_{1 \leq u \leq v \leq n} \left\{ \sum_{j=1}^u p_{i_j 1} + \sum_{j=u}^v p_{i_j 2} + \sum_{j=v}^n p_{i_j 3} \right\}$$

a formulation we encountered in 4.3.4..

Exchanging i_k and i_{k+1} , we find that only H_k, H_{k+1}, K_k and K_{k+1} are changed into $H'_k, H'_{k+1}, K'_k, K'_{k+1}$. The old sequence will be better than the new one if

$$\begin{aligned} & \max (H_{k+1} + \max_{1 \leq u \leq k+1} \{K_u\}, H_k + \max_{1 \leq u \leq k} \{K_u\}) \\ & \leq \max (H'_{k+1} + \max \{K_1, \dots, K_{k-1}, K'_k, K'_{k+1}\}, \\ & \quad H'_k + \max \{K_1, \dots, K_{k-1}, K'_k\}) \end{aligned} \quad (*)$$

We can draw no general conclusions now. However, if

$$\max_{u \leq v} K_u = K_v$$

which is the case when

$$\min_k \{p_{k1}\} \geq \max_k \{p_{k2}\}$$

then (*) reduces to:

$$\max (H_{k+1} + K_{k+1}, H_k + K_k) \leq \max (H'_{k+1} + K'_{k+1}, H'_k + K'_k)$$

leading easily to:

$$\begin{aligned} & \min (p_{i_k 1} + p_{i_k 2}, p_{i_{k+1} 3} + p_{i_{k+1} 2}) \leq \\ & \quad \min (p_{i_{k+1} 1} + p_{i_{k+1} 2}, p_{i_k 3} + p_{i_k 2}) \end{aligned}$$

Comparison with theorem 4.3.1.A. shows, that in this case the $n|3|F|F_{\max}$ problem is a special case of the $n|2|F|F_{\max}$ problem with processing times:

$$p'_{i_k 1} = p_{i_k 1} + p_{i_k 2}$$

$$p'_{i_k 2} = p_{i_k 3} + p_{i_k 2}$$

so that Johnson's algorithm produces the optimal sequence.

Szwarc [99] tries to develop a comparable method, applicable to more general cases. However, his proofs are incorrect, as shown by Arthanari and Mukhopadhyay [1]. These authors also give solutions to two more special cases:

$$(1) \quad \max_k \{p_{k1}\} \leq \min_k \{p_{k2}\}$$

In this case, we have:

$$\max_{1 \leq u \leq v} \{K_u\} = K_1 = p_{i_1 1}$$

so:

$$\sum_{j=1}^n Y_{i_j} = p_{i_1 1} + p_{i_1 2} - p_{i_1 3} + \max(p_{i_1 3}, \max_{2 \leq v \leq n} I_v)$$

where

$$I_v = \sum_{j=1}^v p_{i_j 2} - \sum_{j=2}^{v-1} p_{i_j 3}$$

For $i_1 = 1, \dots, n$, we can find $\min \{ \max_{2 \leq v \leq n} \{I_v\} \}$ by Johnson's algorithm, and thereby solve this problem.

$$(2) \quad \max_k \{p_{k3}\} \leq \min_k \{p_{k2}\}$$

In this case, $\max_v \{H_v\} = H_n$; this problem is then again solved by solving $n | 2 | F | F_{\max}$ problems.

Apart from the integer programming formulation by Wagner [102], given in 3.3., there have been several attempts to solve the $n | 3 | F | F_{\max}$ problem by branch-and-bound methods. Ignall and Schrage [52] use a "frontier search" algorithm, where branching is done by successively scheduling next all the jobs that are yet unscheduled. A lower bound LB is calculated as follows. Let T_1, T_2, T_3 be finishing times of the set J_s of scheduled jobs on M_1, M_2, M_3 , $\bar{J}_s = \mathcal{J} - J_s$.

$$LB = \max \begin{cases} T_1 + \sum_{J_s} p_{k1} + \min_{\bar{J}_s} \{p_{k2} + p_{k3}\} \\ T_2 + \sum_{J_s} p_{k2} + \min_{\bar{J}_s} \{p_{k3}\} \\ T_3 + \sum_{J_s} p_{k3} \end{cases}$$

This lower bound is not very sophisticated and computing results are not very impressive, although computation time is reduced by applying a simple dominance criterium whereby some nodes can be eliminated directly.

A similar method has been developed by Lomnicki [63] and subsequently been extended by Brown and Lomnicki [19] to cover the $n | m | P | F_{\max}$ problem. Their bound is the following one. Suppose J_{i_1}, \dots, J_{i_k} have been scheduled so far, and J_{i_k} is finished on M_ℓ at $T_{k\ell}$. Then define:

$$g_\ell = T_{k\ell} + \sum_{j=k+1}^n p_{i_j \ell} + \min_{k+1 \leq j \leq n} \left\{ \sum_{r=\ell+1}^m p_{i_j r} \right\}$$

Then $g = \min_{1 \leq \ell \leq m} \{g_\ell\}$ gives a lower bound at every node. Lower bounds for the first n nodes can be developed likewise. The bounds are identical to those of Ignall and Schrage if $m = 3$.

A few other branch-and-bound methods for the $n|m|P|F_{\max}$ problem, that could be applied here, are mentioned by Bakshi and Arora [6]; we will not go into them here any more.

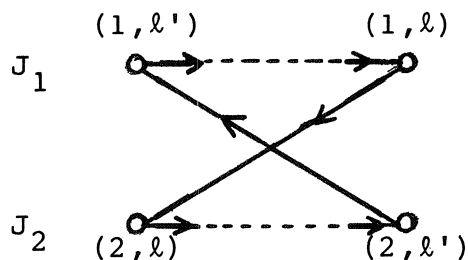
4.5. The two-jobs problem ($2|m$)

The highly artificial $2|m|F|F_{\max}$ and $2|m|G|F_{\max}$ problems are being considered here, because there are two interesting approaches to it that might find wider application. We shall follow in this section the convention by which M_ℓ means that $(J_1, M_\ell) < (J_2, M_\ell)$ and \bar{M}_ℓ means that $(J_2, M_\ell) < (J_1, M_\ell)$.

In the first place, we note that infeasible sequence here is characterized by containing $M_\ell \bar{M}_{\ell'}$, while in the technological machine ordering we find:

$$(J_1, M_{\ell'}) < (J_1, M_\ell)$$

$$(J_2, M_\ell) < (J_2, M_{\ell'})$$



Furthermore, it is easy here to distinguish sequences that can never be optimal. We have the following rule (developed by Akers and Friedman):

if the following orders are prescribed:

$$(J_1, M_\ell) \ll (J_1, M_{\ell'}) \ll \dots \ll (J_1, M_{\ell''})$$

$$(J_2, M_\ell) \ll \dots \ll (J_2, M_{\ell'}) \ll (J_2, M_{\ell''})$$

then disregard any sequence containing $M_\ell \bar{M}_\ell, M_{\ell''}$.

From this rule more specific rules (eight in all) may be derived by interchanging J_1 and J_2 , and by disregarding $M_\ell, M_{\ell''}$ or both. In this (non-numerical) way, one can delimit the search for an optimum to a smaller set of feasible sequences.

Hardgrave and Nemhauser have developed a graphical technique to solve the $2|m|G|F_{\max}$ problem. We shall illustrate this technique by a $2|4|G|F_{\max}$ example. Suppose the technologically prescribed machine orderings are

$$(J_1, M_1) \ll (J_1, M_3) \ll (J_1, M_2) \ll (J_1, M_4)$$

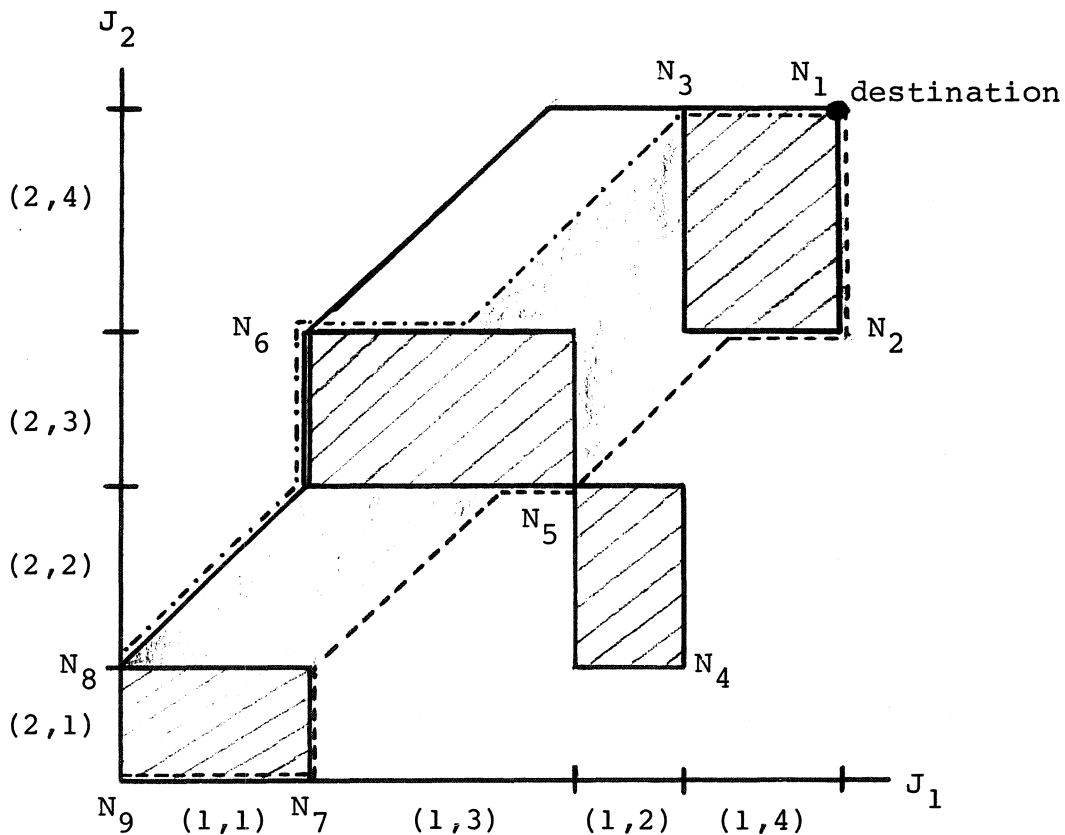
$$(J_2, M_1) \ll (J_2, M_2) \ll (J_2, M_3) \ll (J_2, M_4)$$

Processing times are:

$$p_{11} = 5 \quad p_{12} = 3 \quad p_{13} = 7 \quad p_{14} = 4$$

$$p_{21} = 3 \quad p_{22} = 5 \quad p_{23} = 4 \quad p_{24} = 6.$$

We can depict this information in the diagram below, taking one axe for each job.



The lines represent a feasible schedule: they run from $(0,0)$ to $(\sum p_{1k}, \sum p_{2l})$, and avoids the hatched areas (because that would imply a machine ran on two jobs simultaneously). F_{\max} is equal to the sum of vertical (horizontal) segments plus $\sum p_{k1}$ ($\sum p_{k2}$).

In principle, one would have to draw all 2^m lines and decide which one has the smallest sum of vertical segments. However, one can avoid some schedules by drawing two from the origin and two from the destination that favour consistently job 1 or job 2. The dotted line for instance favours job 1 consistently. All potentially optimal schedules then have to lie in the area, formed by the intersection of these four schedules. This area has been shaded in the drawing. We see then that the schedule marked by ... is optimal with $F_{\max} = 26$.

Szwarc [99] evaluates the lengths of all paths in the shaded area by dynamic programming. He considers the origin, the destination and the north-west and south-east corner of each rectangle as nodes N_j ($j = 1, \dots, J$). Nodes are ordered lexicographically by decreasing (x, y) coordinates. Define the (vertical) distance between N_i and N_j (where $i > j$):

$$d(N_i, N_j) = \max((y_j - y_i) + (x_j - x_i), 0).$$

Now, define $\pi(N_i)$ to be the set of N_j ($j > i$) with a feasible path to N_i that does not contain any other N_j (e.g., $\pi(N_2) = \{N_4, N_5, N_6\}$) and define

$$f(N_i) = \min_{\pi(N_i)} (d(N_i, N_j) + f(N_j)).$$

Taking $f(N_1) = 0$ (N_1 is the destination), $f(N_J)$ will give the minimum value of F_{\max} .

Szwarc suggests that it may sometimes be possible to solve a $n|m|G|F_{\max}$ problem by first solving $\binom{n}{2}$ $2|m|G|F_{\max}$ problems and combining the solutions. This method obviously cannot guarantee a feasible solution.

5. The general flow shop and job shop problem

5.1. Introduction

If we now turn finally to the general $n|m|p$, $n|m|F$ and $n|m|G$ problems, the lack of theoretical and practical results becomes particularly obvious. To start with, the only criterium considered here is a minimizing F_{\max} . In the $n|m|P|F_{\max}$ problem, where we only have to consider $n!$ permutation schedules, there are a few theoretical results that reduce the search for the optimum solution. With regards to the $n|m|F|F_{\max}$ problem the common machine order for all jobs does not give much extra information at all, except for theorems 4.3.A. and 4.3.B.. So one might as well study the $n|m|G|F_{\max}$ problem, which still remains the most difficult of them all. Practically no theoretical progress has been made here, but some sophisticated branch-and-bound methods have been developed. However, it looks as though even they cannot solve problems of any appreciable size. If one considers moreover the seeming unrealisticness of the $n|m|G|F_{\max}$ problem in general, things look bleak indeed. It is pretty obvious that present combinatorial methods are not powerful enough to solve these very large problems; the combinatorial proofs encountered so far in simpler problems are already very complicated.

Altogether it is not at all surprising, as mentioned before, that known applications of scheduling theory have mostly been heuristic ones. Below, we shall review the work done so far (as far as this has not been mentioned earlier in chapters 3 and 4), and hopefully await better times.

5.2. The $n|m|P|F_{\max}$ problem

The restriction that "passing is not permitted" reduces the search for an optimum to $n!$ sequences only. The restriction itself is not very realistic; however, $n|m|P|F_{\max}$ solutions

have some heuristic value as well if one regards them as approximations to given $n|m|F|F_{\max}$ problems. As we shall see below, an optimal solution to the latter problem may well be one where "passing" is necessary, but a reasonable solution is better than no solution at all.

Having already mentioned Wagner's integer programming approach [102] and the branch-and-bound method of Brown and Lomnicki [19], we want to pay attention now to theoretical results obtained by Szwarc [100]. These methods allow a sizeable reduction of the search for an optimal $n|m|P|F_{\max}$ solution.

By definition, there are only $n!$ different schedules to consider, each one characterized by a permutation of $(1, \dots, n)$. This is the situation in which combinatorial-analytical methods might indeed be used profitably. The general method to use is familiar by now: eliminate as many sequences as possible and search sensibly among the remaining ones.

What we want to do essentially in the $n|m|P|F_{\max}$ situation is to find some criterium which, if it holds, allows one to eliminate a set of sequences that can never be optimal (or, weaker: to eliminate a set of sequences so that the remaining set contains at least one optimal solution).

All criteria mentioned in Szwarc's article have the same form: if a certain condition $C(\sigma, J_a, J_b)$ holds with regard to a given sequence of jobs σ and jobs J_a and J_b , then we can eliminate all sequences beginning with σJ_b .

Of course, we have to check if there is at least one optimal sequence remaining after the elimination. A way to do this is given by the following criterium.

Suppose π' and π'' are sequences satisfying: $\pi' \cap \pi'' = \emptyset$,
 $(\pi' \cup \pi'') \cap (\sigma J_a J_b) = \emptyset$, $\pi' \cup \pi'' \cup (\sigma J_a J_b) = \{J_1, \dots, J_n\}^*$.
 Then if $C(\sigma, J_a, J_b)$ implies:

$$t(\sigma J_a J_b \pi' \pi'', m) \leq t(\sigma J_b \pi' J_a \pi'', m) \dots \dots \dots (*)$$

(where $t(\pi, \ell)$ is the finishing date of a given sequence π on M_ℓ),
 then we can be since that there is an optimal sequence not
 starting with σJ_b (because we would not increase F_{\max} by moving
 J_a between σ and J_b).

Now Szwarc mentions five of these elimination criteria, four
 of which were known already. First he defines:

$$\Delta_\ell = t(\sigma J_a J_b, \ell) - t(\sigma J_b, \ell).$$

The five criteria now read: eliminate all sequences beginning
 with σJ_b if:

(1) $t(\sigma J_a J_b, \ell) \leq t(\sigma J_b J_a, \ell)$ ($\ell = 2, \dots, m$)
 (due to Dudek and Teuton [27]);

(2) $\Delta_{\ell-1} \leq p_{a\ell}$ ($\ell = 2, \dots, m$)
 $t(\sigma J_a, \ell-1) \leq t(\sigma J_b, \ell-1)$
 (due to Smith and Dudek [95]);

(3) $\Delta_\ell \leq p_{a\ell}$ ($\ell = 2, \dots, m$)
 (due to Bagga and Chakravarti [4]);

(4) $\Delta_{\ell-1} \leq \Delta_\ell \leq p_{a\ell}$ ($\ell = 2, \dots, m$)
 (due to Szwarc);

(5) $\Delta_{\ell-1} \leq p_{a\ell}$ ($\ell = 2, \dots, m$)
 (due to Smith and Dudek [94]).

*) In this context we regard sequences as sets by forgetting
 about precedence relations.

First, we want to check that Szwarc's criterium is really a valid one. To get the flavour of the type of reasoning needed here, we shall follow the proof that Szwarc's criterium (4) is valid in the sense that if it holds, (*) must also be true.

We need two fairly general lemma's:

Lemma 5.2.A.: if (4) holds, then, for any sequence π such that $\sigma \cap \pi = \emptyset$, $J_a \notin \pi$, $J_b \notin \pi$:

$$t(\sigma J_a J_b \pi, \ell) - t(\sigma J_b \pi, \ell) \leq t(\sigma J_a J_b, \ell) - t(\sigma J_b, \ell).$$

Proof: induction on ℓ and p (the number of elements in π).
Trivial for $\ell = 1$ and $p = 1$.

Now: $\ell - 1 \Rightarrow \ell$ ($p = 1$, $\pi = J_s$)

$$\begin{aligned} & t(\sigma J_a J_b J_s, \ell) - t(\sigma J_b J_s, \ell) = \\ & \max(t(\sigma J_a J_b J_s, \ell-1), t(\sigma J_a J_b, \ell)) + p_s \ell \\ & - \max(t(\sigma J_b J_s, \ell-1), t(\sigma J_b, \ell)) - p_s \ell \leq * \\ & \max(t(\sigma J_a J_b J_s, \ell-1) - t(\sigma J_b J_s, \ell-1), \\ & t(\sigma J_a J_b, \ell) - t(\sigma J_b, \ell)) \leq \\ & \max(t(\sigma J_a J_b, \ell-1) - t(\sigma J_b, \ell-1), \\ & t(\sigma J_a J_b, \ell) - t(\sigma J_b, \ell)) = \\ & t(\sigma J_a J_b, \ell) - t(\sigma J_b, \ell) \end{aligned}$$

(the two last steps being justified by the induction step and (4)).

*) $\max(A, B) - \max(C, D) \leq \max(A-C, B-D)$.

We can now repeat the proof for any p , if we assume the case $p - 1$ has been proved.

Lemma 5.2.B.: let ε and ε' be different permutations of the same set of jobs, and π any permutataon, such that $\varepsilon \cap \pi = \emptyset$. Then:

$$t(\varepsilon, \ell) < t(\varepsilon', \ell) \Rightarrow t(\varepsilon\pi, \ell) < t(\varepsilon'\pi, \ell)$$

Proof: the proof is by induction and based again on

$$t(\varepsilon J_s, \ell) = \max(t(\varepsilon J_s, \ell-1), t(\varepsilon, \ell)) + p_{s\ell} \quad (**)$$

Now, we can prove that (4) implies (*): if (4) holds, then by lemma 5.2.A.:

$$\begin{aligned} t(\sigma J_a J_b \pi', \ell) - t(\sigma J_b \pi', \ell) &\leq \\ t(\sigma J_a J_b, \ell) - t(\sigma J_b, \ell) &= \Delta_\ell \end{aligned}$$

By (4) $\Delta_\ell \leq p_{a\ell}$, so (by (**)):

$$t(\sigma J_a J_b \pi', \ell) \leq t(\sigma J_b \pi', \ell) + p_{a\ell} \leq t(\sigma J_b \pi' J_a, \ell)$$

Lemma 5.2.B. (with $\varepsilon = \sigma J_a J_b \pi'$, $\varepsilon' = \sigma J_b \pi' J_a$, $\pi = \pi''$, $\ell = m$) now gives (*) immediately.

So we know (4) is a valid criterium. What about the other ones? (1) is known to be false; Karush [56] already provided a counter example. Szwarc himself has given in an earlier article a counter example to (3). He now gives a counter example to show that application of criterium (5) at least does not imply (*). So we are left with (2) and (4). By a complicated proof similar to the one above Szwarc now shows that (2) implies (4); therefore (2) also is a valid criterium. Nevertheless (4)

is a stronger one, because any sequence eliminated by (2) could have been eliminated by (4), but not vice versa!

We now give a small example to illustrate the use of Szwarc's criterium.

Suppose $n = 3$ and $m = 3$. Let:

$$p_{11} = 1 \quad p_{12} = 2 \quad p_{13} = 3$$

$$p_{21} = 2 \quad p_{22} = 1 \quad p_{23} = 2$$

$$p_{31} = 1 \quad p_{32} = 3 \quad p_{33} = 3$$

(1) Taking $\sigma = \emptyset$ in (4), we see that we have:

$\Delta_1 = p_{a1} \leq \Delta_2 \leq \dots \leq \Delta_\ell$ and because $\Delta_\ell \leq p_{a\ell}$: $p_{a1} \leq p_{a\ell}$ ($\ell = 2, \dots, m$). So in the example J_1 and J_3 could play the role of J_a . First, take $J_a = J_1$, $J_b = J_2$ (σ is still \emptyset).

We have to check:

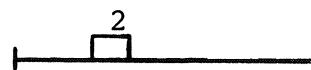
$$\Delta_1 \leq \Delta_2 \leq p_{12} = 2$$

$$\Delta_2 \leq \Delta_3 \leq p_{13} = 3$$

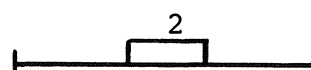
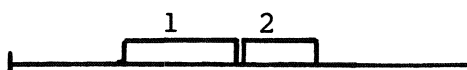
Draw up two Gantt charts



$$\Delta_1 = 1$$

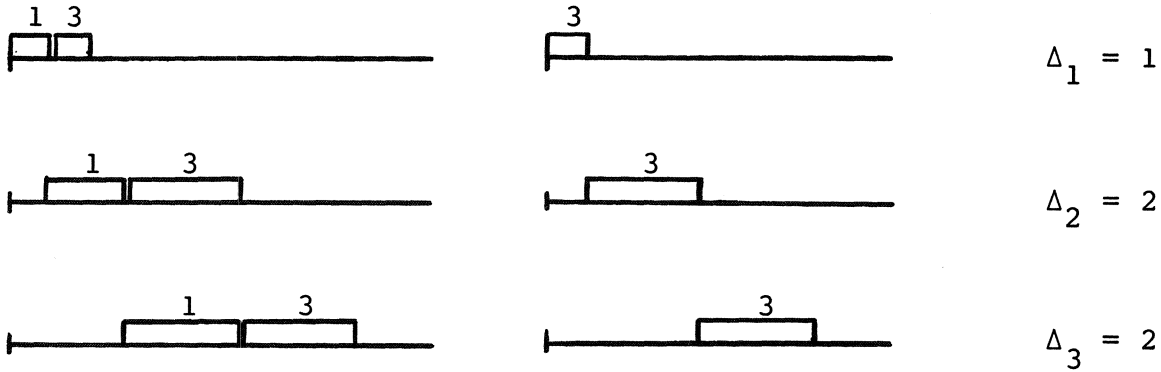


$$\Delta_2 = 1$$



$$\Delta_3 = 3$$

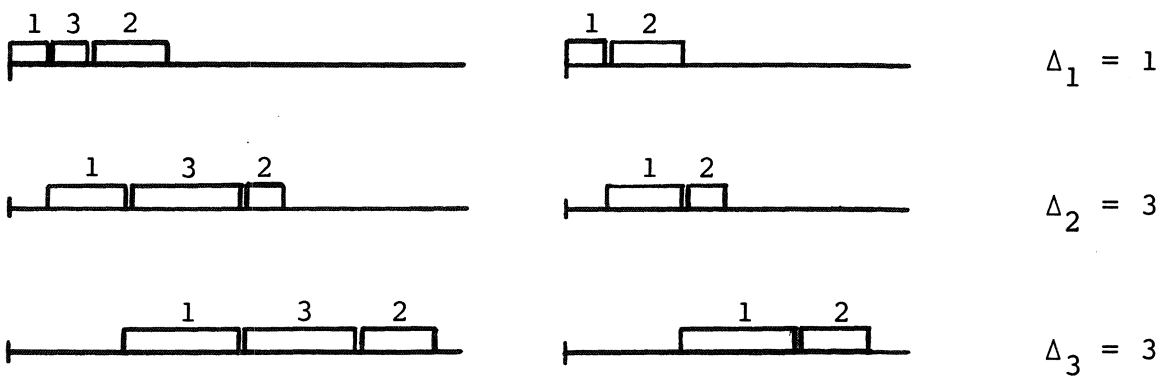
Both inequalities hold. J_2 cannot be first.
 Now take $J_a = J_1$ and $J_b = J_3$.



Again the inequalities hold. So J_3 cannot be first too, and J_1 must be first.

Now we try to eliminate jobs from the last position, by filling up a schedule back to front. For job J_a it then has to be true that $p_{am} \leq p_{ak}$ ($k = 1, \dots, m-1$). There is no job satisfying those constraints.

(2) We know J_1 must be first. So we take $\sigma = J_1$. We try J_3 first as J_a , and J_2 as J_b .

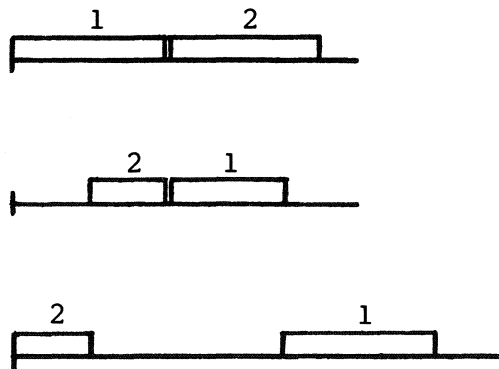


Again we find: $\Delta_1 \leq \Delta_2 \leq p_{32} = 3$ and $\Delta_2 \leq \Delta_3 \leq p_{33} = 3$. So we can eliminate a sequence, starting with (1,2), which leaves only (1,3,2) as the optimal sequence with $F_{\max} = 11$.

The unfortunate thing is that Szwarc neither gives an algorithm based on this criterium nor any computing experience with it. Claiming he does this "intentionally", one wonders about the goal he is trying to attain here. Still, it should not be too difficult to devise a branch-and-bound method to search the best among the remaining sequences. Moreover, Szwarc himself hints that it may be possible to find even sharper elimination criteria.

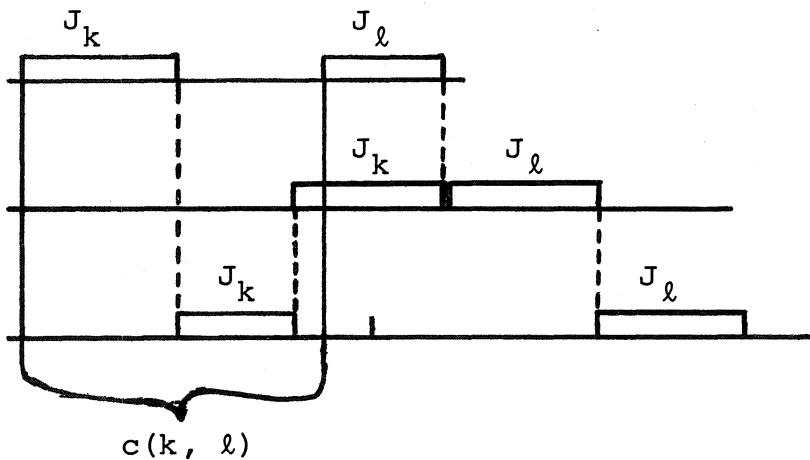
Having already mentioned the existence of several branch-and-bound methods to solve the $n|m|P|F_{\max}$ problem (see Bakshi and Arora [6]), our discussion would have to end here, were it not that under very special circumstances a $n|m|F|F_{\max}$ problem degenerates into a $n|m|P|F_{\max}$ problem. We are referring to the case in which no intermediate storage is allowed (this implies (J4) is no longer valid), so that all operations have to be processed directly after each other.

This problem has been attacked by Wismer [103] and by Reddi and Ramamoorthy [82] practically simultaneously. Although Wismer is rather vague on this point, his method also depends on the fact that each machine processes all jobs in the same order. (A sequence like the one below would not be allowed,



although all operations are performed without delay). Reddy and Ramamoorthy deal with a F and therefore P situation straight from the beginning.

Now, it is obvious that the minimum time between initiation of J_k and the initiation of J_l is a function c of k and l only. It is not difficult to derive an exact formulation for this minimum time, but the easiest way to conceive of this function is to picture Gantt charts for J_k and J_l , and, fixing the one for J_k , to move the one for J_l as far to the left as possible until two operations "touch" each other.



If we introduce a job J_0 with

$$c(0, k) = 0 \quad (k = 1, \dots, m)$$

$$c(k, 0) = \sum_{j=1}^m p_{kj} \quad (k = 1, \dots, m)$$

we see that the minimization of F_{\max} is equal to the minimization of

$$\sum_{j=0}^m c(i_j, i_{j+1}) + c(i_m, i_0)$$

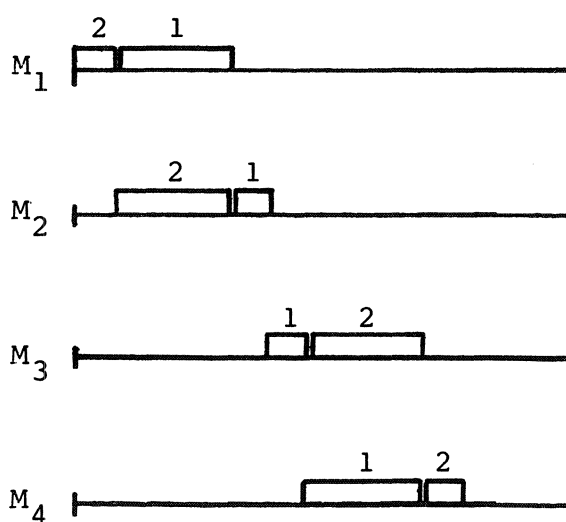
over all permutations (i_0, \dots, i_m) of $(0, \dots, m)$. This is easily recognized as another example of the Travelling Salesman Problem.

Examples of a process where intermediate storage would indeed not be allowable can be found by looking at steel mills or at computers processing a set of jobs.

5.3. The $n|m|F|F_{\max}$ problem

We now turn to the general flow shop problem. Generally speaking all $(n!)^m$ possible sequences are feasible (i.e. compatible with the given machine ordering per job) in this situation, as is easy to prove. However, theorem 4.3.B. permits elimination of those sequences with different orderings on the first and second, or $(m-1)^{\text{th}}$ and m^{th} machine, leaving $(n!)^{m-2}$ to be evaluated - a considerable number.

One might hope that the optimal sequence was one whereby jobs did not pass each other. However, the example below for $n = 4$ shows that this is not the case; the depicted sequence is optimal with respect to F_{\max} .



It might be interesting to find out what percentage of random $n|m|F|F_{\max}$ problems has a "non-passing" optimal solution.

No more specific theory for the general flow shop problem exists. Apart from the special cases treated in chapter 4, it is just as difficult as the general job shop problem to which we turn now.

5.4. The $n|m|G|F_{\max}$ problem

There is little doubt that we have now arrived at the most formidable problem of them all. As Conway, Maxwell and Miller [24] put it discouragingly: "Many proficient people have considered this problem, and all have come away essentially empty-handed. Since this frustration is not reported in the literature, the problem continues to attract investigations, who just cannot believe that a problem so simply structured can be so difficult, until they have tried it."

Throughout the report, methods to attack this problem have been mentioned. We have noted the failure so far of integer programming methods and the lack of any combinatorial-analytical results. Also we have introduced the concepts of active and non-delay schedules; though we can easily generate all active schedules (e.g. by the algorithm of Giffler and Thompson [39]), this class is still too large to be completely enumerated within reasonable time. There are mainly two things left to do. We shall take a look at methods to find infeasible solutions, and we shall review attempts to solve this problem by branch-and-bound methods. Throughout this section we rely rather heavily on the disjunctive-graph model formulated in 2.1..

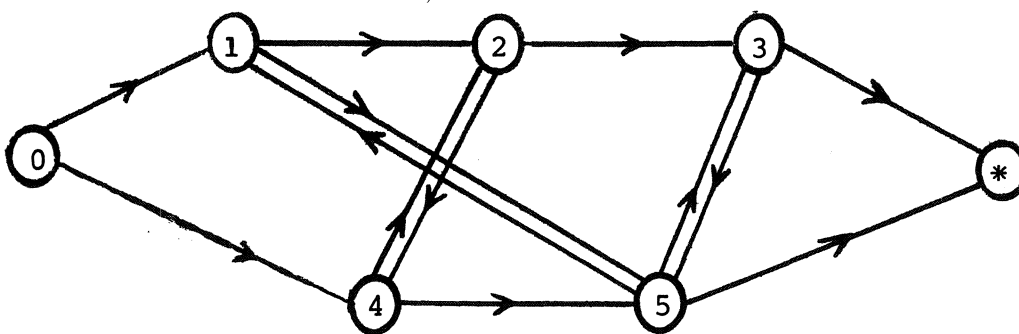
5.4.1. Elimination of infeasible sequences

Unlike the flow shop problem, some solutions to the $n|m|G$ problem may be infeasible. Already (in 2.1.) we have seen that these infeasible solutions correspond to cycles in a directed graph.

To detect these we can use a simple algorithm, like the one developed by Marimont (reported in [6]).

First we number all operations from 1 to nm , starting with those of J_1 , etc. ^{*)}. (We stick to this convention throughout this section.) Then we construct a $(nm \times nm)$ -matrix of which the i - j^{th} entry is 1 if operation i directly precedes operation j (by technological reasons or by the proposed solution), and 0 otherwise. Any operation with an empty row or column can be scheduled and removed. If all operations can be removed, the solution is feasible.

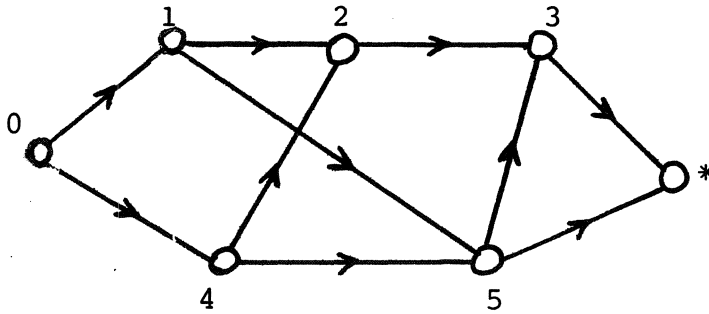
Example: Suppose we have a $2|2|G|F_{\max}$ problem, corresponding to the following disjunctive graph ^{**)}:



*) All methods in this section are also applicable if any job J_k does not pass all machines or passes some machines twice.

***) See the footnote above.

One proposed solution might be:



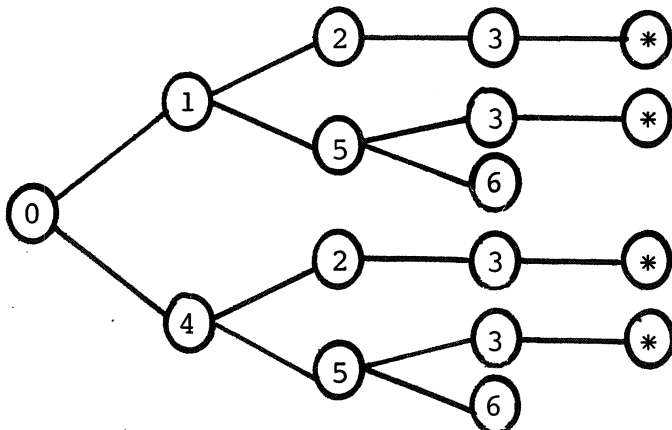
The matrix is:

	0	1	2	3	4	5	6
0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0
2	0	0	0	1	0	0	0
3	0	0	0	0	0	0	1
4	0	0	1	0	0	1	0
5	0	0	0	1	0	0	1
6	0	0	0	0	0	0	0

The solution is obviously feasible.

Baskshi and Arora ([6]) and Ashour ([2]) mention another trivial technique to eliminate infeasible solutions, developed by Nelson. We draw up a tree by starting with operation 0, and branching to every node that directly follows the present one. This process does not terminate if the solution is infeasible.

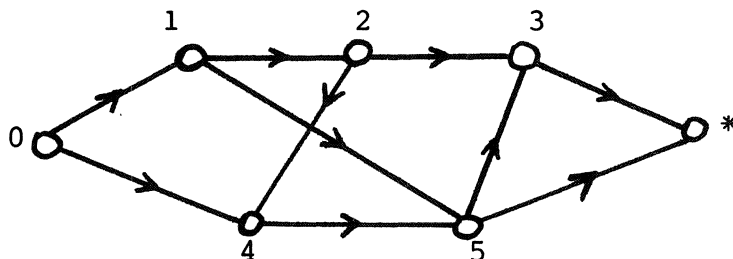
For our example, the tree would look like below.



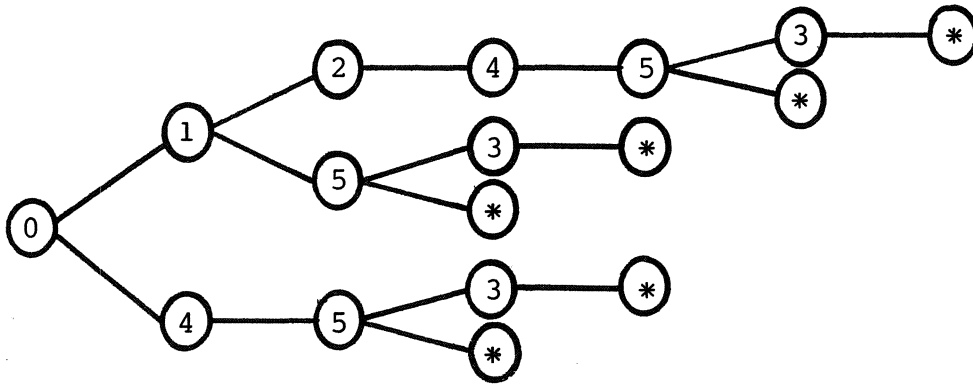
This method is slightly more interesting than the previous one, because, if we assign length p_i to branch $(i-j)$, the longest branch in this tree will be equal to F_{\max} (this is again equivalent to saying that F_{\max} is equal to the length of the longest (the so-called critical) path in the graph).

Now this implies that, if the tree corresponding to some solution, is contained in the tree corresponding to another one, the latter solution can never be optimal.

For instance, suppose we have the following solution, where one disjunctive arc has been changed to another direction:



The corresponding tree is:



and by comparing it to our former tree, we see that this solution can never be optimal.

Now, here we have a non-numerical technique to detect potentially optimal sequences. What is far more important, however, is the following. We have now seen that, if in a disjunctive graph we assign a direction to the disjunctive arcs in accordance with some proposed solution, the maximum flow time of this solution is equal to the longest (critical) path in the created directed graph. (Any infeasible solution will lead to loops.)

This insight has led to the best of the branch-and-bound methods, that we shall now deal with.

5.4.2. Branch-and-bound methods

In order to facilitate discussions, we first restate more formally the disjunctive graph model.

The disjunctive graph G , corresponding to a given $n|m|G|F_{\max}$ problem, is completely characterized by three sets \mathcal{N} , \mathcal{C} and \mathcal{D} .

\mathcal{N} is the set of nodes of G , each node corresponding to an operation. We index these nodes by first taking the n_1 operations of $J_1^{*})$ and number them in the given machine order $1, \dots, n_1$.

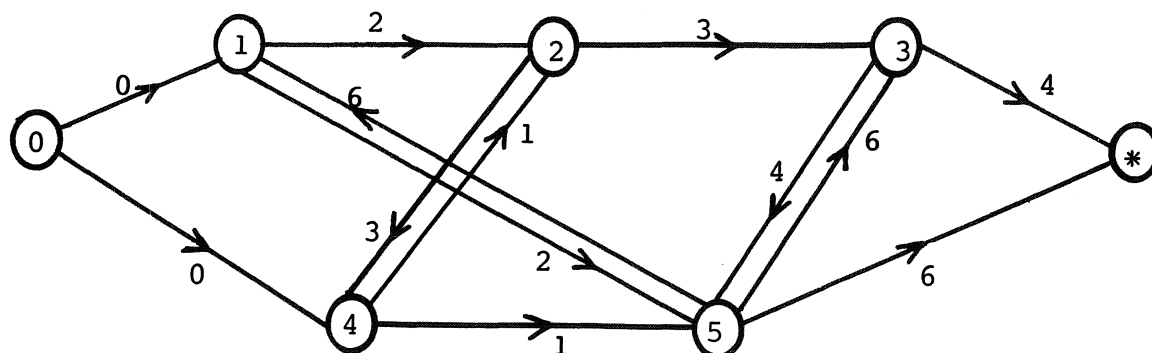
*) We may drop the assumption that $n_1 = n$, etc..

Likewise, the n_2 operations of J_2 are numbered n_1+1, \dots, n_2 , etc.. Also included in \mathcal{N} are two dummy operations 0 and *, whose meaning will be clear later on. We call the set of the first operations of J_1, \dots, J_n α , and the set of the last operations β . Furthermore we designate by μ_ℓ all the operations that are performed on machine M_ℓ ($\ell = 1, \dots, m$).

\mathcal{C} is the set of all conjunctive arcs. These are ordinary directed arcs that connect node k to node $(k+1)$ ($k = 1, \dots, n_1-1, n_2, \dots, n_2-1, n_3, \dots, n_n-1$), signifying that k directly precedes $(k+1)$ for technological reasons. Furthermore there are n conjunctive arcs from 0 to the nodes in α , and n conjunctive arcs from β to *. To any of these arcs we assign a length p_k , corresponding to the duration of the operation that the arc is branching from (take $p_0 = 0$).

\mathcal{D} is the set of disjunctive arcs. Any disjunctive arc can be thought of as a pair of oppositely directed arcs, each with a length assigned to it according to the rule above, that connect all pairs of operations from different jobs in μ_ℓ ($\ell = 1, \dots, m$).

Below is the former example; job 1 consists of operations 1, 2, 3 with processing times 2, 3, 4; job 2 consists of operations 4, 5 with processing times 1, 6.



Now, by "resolving a disjunctive arc" we shall mean choosing one directed arc and (temporarily) dropping the other. This corresponds to assigning precedence to one of the two operations on the machine under consideration. If we resolve a number of disjunctive arcs, forming a subset D of \mathcal{D} , we shall speak of a partial solution. Associated with any partial solution is a set $N \subset \mathcal{N}$, containing the nodes all of whose disjunctive arcs have been settled. When evaluating any partial solution, we will usually disregard any non-resolved disjunctive arcs. When all disjunctive arcs have been resolved, we have a solution to the problem, that is feasible if the now-created directed graph does not contain any loops. The value of F_{\max} for this particular solution then corresponds to the length of the longest path in this directed graph. A very efficient algorithm, devised by Kelley, exists for calculation of this so-called critical path CP. Basically it uses the formula

$$\begin{cases} CP(k) = \max \{CP(j) + p_j\} \\ CP(0) = 0 \end{cases}$$

the maximum being taken over all nodes directly-preceding node k , and the length of CP being given by $CP(*)$.

Several algorithms either implicitly or explicitly depend on the above model. We shall distinguish two main groups, and also pay attention to Balas's algorithm ([8]), which is mainly of theoretical interest.

The first group consists of the algorithms of Greenberg, Nemeti, Charlton and Death, Nabeshima and Sussmann. In fact, Greenberg ([43]) was one of the first authors to apply branch-and-bound techniques to the scheduling problem. Essentially, he first disregards all disjunctive arcs and then adds them one by one

in an unspecified order, branching by resolving them either in one or in the other direction. At each branch a lower bound is given by the longest path in the graph constructed so far^{*)}. Using a frontier search method, gradually the optimal solution is built up.

Now this method obviously is not very efficient. There is just no need to resolve all disjunctive arcs in this way, because very often two operations on the same machine will not be competing for time at all. Only for those operations that do have this conflict, we need to settle the disjunctive arcs one way or the other.

The above consideration has led to the practically equivalent algorithms of Nemeti [77], Charlton and Death [21], Sussmann [98] and Nabeshima [76]. Again we start by disregarding all disjunctive arcs. Then, by calculating the present earliest starting times t_k of all operations by Kelley's algorithm, we look if there is at present any conflict between two operations on one machine (i.e., $t_j - t_k < p_k$ and $t_k - t_j < p_j$ for $j, k \in \mu_\ell$)^{**)}. If not, we have a complete solution. Otherwise, we select a conflict in a heuristic way (several recipes for this are given) and branch by resolving the corresponding disjunctive arc in one way or the other. Again, a lower bound is given by the longest path in the graph constructed so far. Proceeding either by newest active node (Charlton and Death, Sussmann) or frontier search (Nemeti), we arrive at the optimal solution.

It is interesting to notice that no infeasible solutions are ever generated this way, since any path existing from j to k or vice versa prior to the resolution of the disjunctive arc

*) Infeasible solutions are quickly discovered by this value becoming infinitely large.

***) t_j is the starting date of operation j .

must have had a length of at least either p_j or p_k , in which case there would be no conflict. However, the argument does not apply any more if sequence-dependent set-up times are included in the p_k 's. Nabeshima ([76]) gives a counter-example to show this.

The above bound is improved by Charlton and Death in a later article ([22]). For any partial solution with corresponding sets $D \subset \mathcal{N}$ and $N \subset \mathcal{N}$, they take the maximum of the longest path and

$$\max_{\ell} \left\{ \max_{j \in N \cap \mu_{\ell}} \{t_j + p_j\} + \sum_{\substack{j \in (\mathcal{N}-N) \\ \cap \mu_{\ell}}} p_j \right\}$$

Nabeshima finally has stressed the potential usefulness of this algorithm for other criteria; computation of the lower bounds is not so simple then, however.

Although computing experience with some of the above algorithms is not at all bad, the lower bounds are just not very sharp. To see how they might be increased, we turn to the second group where we find the work of Brooks and White, Schrage, Florian, Trepan, McMahon, Bratley and Robillard.

Brooks and White ([18]) in the first branch-and-bound solution to the scheduling problem essentially propose the following algorithm. For each partial solution consider the set S_0 of scheduleable operations (i.e. the successors to N ; in the first step of the algorithm, take $S_0 = \alpha$). Now find operation k in S_0 so that

$$t_k + p_k = \min_{j \in S_0} \{t_j + p_j\}$$

Suppose k is performed on M_ℓ . We then have a conflict between operation k and all other scheduleable operations on M_ℓ ^{*)}. We branch by successively scheduling first all operations in $S_0 \cap \mu_\ell$. For each branch a lower bound is computed in the following way. First we find the set \mathcal{M}^* of all machines that perform at least one final operation (i.e. those machines M_ℓ for which $\mu_\ell \cap \beta \neq \emptyset$). Next we calculate the earliest finishing time T_ℓ on each machine $M_\ell \in \mathcal{M}^*$ by disregarding all unresolved disjunctive arcs except on M_ℓ itself, where operations are scheduled according to the FIFO principle (i.e. the operations on M_ℓ are performed in order of increasing earliest starting times t_k). The maximum of the earliest finishing times T_ℓ over all $M_\ell \in \mathcal{M}^*$ then gives a lower bound for the particular branch under consideration.

The above formulation covers the rather vague terminology of Brooks and White (mainly aimed at the $n|m|F|F_{\max}$ case) and also the work of Florian, Trépan and McMahon ([32]). The latter authors' algorithm is already superior to those of Schrage ([89]), who uses a similar approach with less sharp bounds, and Balas ([8]), who will be treated later. However, the lower bounds are not yet completely satisfactory. The restriction to the set \mathcal{M}^* of machines that perform at least one final operation, has been made essentially because we wanted to disregard everything that happened to the jobs after they had passed the machines in \mathcal{M}^* . Obviously this can only be justified if we stick to machines that are in the above sense "final", because otherwise the method would lead to worthless bounds.

Nevertheless, we would like to extend the calculation of the bound to the set $\mathcal{M}_0 \supset \mathcal{M}^*$ of all machines that still have to perform some unscheduled operations. What one could do then (Florian and Sang, [33]) is treat every machine $M_\ell \in \mathcal{M}_0$ as a

*) If k is the only one, just schedule k and go on.

"final" one, and calculate earliest finishing times T_ℓ on each M_ℓ by again disregarding all unsettled disjunctive arcs and FIFO-scheduling on M_ℓ itself. Next we could calculate for each $M_\ell \in \mathcal{M}_0$:

$$Q_\ell = \min_{J_k \in \mu_\ell} \{q_k\}$$

where q_k is the sum of the processing times of all operations remaining for J_k after M_ℓ . Then,

$$\max_{\mathcal{M}_0} \{T_\ell + Q_\ell\}$$

would give a lower bound for the branch in question.

This is still not very satisfying, for, given $M_\ell \in \mathcal{M}_0$, one would rather use q_k directly for scheduling the jobs on M_ℓ . instead of just using FIFO-scheduling and adding q_k afterwards. So, in fact, to calculate the lower bound, we have to solve a number of one machine problems whereby jobs are available on M_ℓ at date t_k^* , take p_k time-units to be processed and then have "tails" q_k remaining before they are finished; the objective is to minimize C_{\max} , including the q_k . Doing this for all machines $M_\ell \in \mathcal{M}_0$ gives us a number of values for C_{\max} , the maximum of which then provides the lower bound.

Now, obviously the usefulness of this lower bound heavily depends on speedily finding the optimum sequence for all these one machine problems. Bratley, Florian and Robillard ([17]) who advocate this approach, have devised an implicit enumeration (**)

*) Formerly called r_k !

**) The formal difference between implicit enumeration and branch-and-bound is that in the former we gradually try to improve a "good" starting solution (using bounds if necessary).

algorithm to solve this $n|1$ problem^{*)}, that looks very much like their algorithm in 4.2.5.

A good initial solution is given by ordering the jobs J_k for this $n|1$ problem according to the following rule: start with J_k with minimal t_k , at any time to choose of the available jobs J_i with $t_i \leq t$ the one with largest q_i , break ties by largest d_i ; if no job is available again take the one with lowest t_k . (Several samples are solved below).

In gradually improving the starting solution, the lower bound becomes important. Let S be the set of scheduled jobs. We have a lower bound then:

$$\max (LB_1, LB_2)$$

where

$$LB_1 = \max_S \{t_k + p_k + q_k\}$$

and

$$LB_2 = \min_{B_p} \{t_k\} + \sum_{B_p} p_k + \min_{B_p} \{q_k\},$$

where B_p is the last block in the given schedule, blocks having been defined previously in 4.2.5.. It is easy to see that LB_2 can be increased by 1 if the last job scheduled is not the one with minimal q_k over B_p .

If in the initial solution J_c is the job with $C_{\max} = a_c + b_c + q_c$, then it is again easy to see that this solution is optimal if

*) We could regard this as a $n|2|F, (J2), (M8)|C_{\max}$ problem by regarding q_k as the processing time on a non-bottleneck M_2 ; this does not seem to lead anywhere.

J_C comes last in its block and has the smallest tail of its block. However, if this is not the case, and if $a_C + b_C + q_C$ is not equal to LB for this solution, we have to find a better one by branching and bounding^{*)}, aided by the following trivial lemma's:

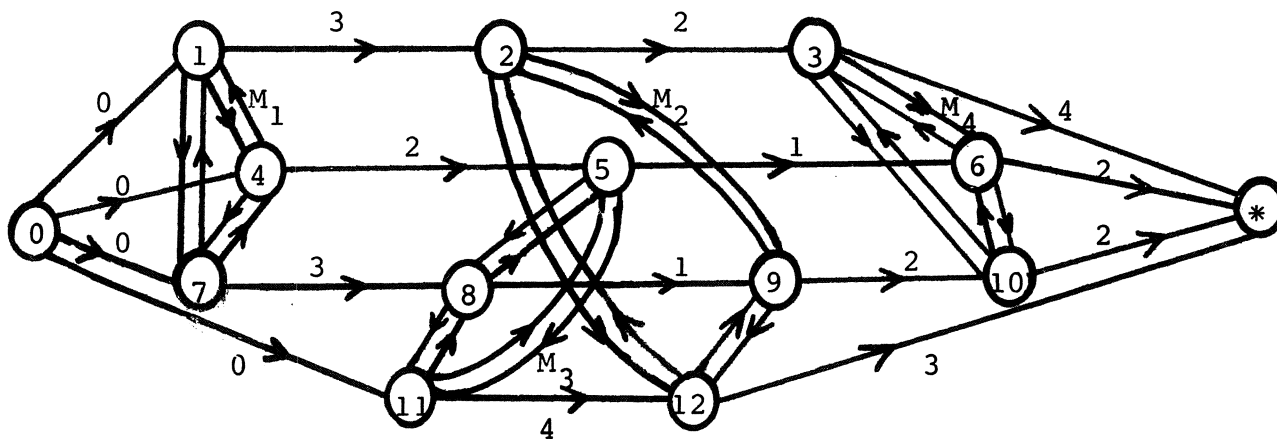
Lemma 5.4.2.A.: if job k could be finished, before job l is available, schedule it.

Lemma 5.4.2.B.: if at any date t , $t_k < t$ for all remaining k , then schedule the remaining jobs by decreasing q_k .

Lemma 5.4.2.C.: any solution can only be improved by moving J_C forward.

Lemma 5.4.2.D.: except in consequence of 5.4.2.C. it is no use backtracking over an unavoidable gap (see lemma 4.2.5.B.).

To show the application of the algorithms in the second group, let us calculate the bounds for one problem situation. The disjunctive graph is given below (the lengths of the disjunctive arcs have not been added, but are clear from the picture).



*) Bounds are recalculated if a gap appears in the schedule.

No jobs have been scheduled so far; $N = \{0\}$. So $S_0 = \{1,4,7,11\}$. Now $t_4 + p_4 = 2$ is minimal over S_0 , so we restrict attention to $\{1,4,7\}$. We have to create three branches.

First, we compute the bounds by the first method. $\mathcal{M}^* = \{M_2, M_4\}$.

(1) Schedule 1 first.

Then on M_2 : $t_2 = 3$ $t_9 = 7$ $t_{12} = 4$

Choose order : 2 - 12 - 9; $T_2 = 10$

On M_4 : $t_3 = 5$ $t_6 = 6$ $t_{10} = 9$

Choose order : 3 - 6 - 10; $T_4 = 13$

Lower bound : $\max(10, 13) = 13$.

(2) Schedule 4 first.

On M_2 : $t_2 = 5$ $t_9 = 6$ $t_{12} = 4$

Choose order : 12 - 2 - 9; $T_2 = 11$

On M_4 : $t_3 = 7$ $t_6 = 3$ $t_{10} = 8$

Choose order : 6 - 3 - 10; $T_4 = 13$

Lower bound : $\max(10, 13) = 13$.

(3) Schedule 7 first.

On M_2 : $t_2 = 6$ $t_9 = 4$ $t_{12} = 4$

Choose order : 9 - 12 - 2; $T_2 = 11$

On M_4 : $t_3 = 8$ $t_6 = 6$ $t_{10} = 6$

Choose order : 10 - 6 - 3; $T_4 = 14$

Lower bound : $\max(10, 14) = 14$.

We note that calculating the lengths of the longest path in the three cases above would have given us lower bounds of 11, 11 and 12 respectively.

Now we extend \mathcal{M}^* to all $\mathcal{M}_0 = \{M_1, M_2, M_3, M_4\}$. The bounds then become:

(1) Schedule 1 first.

$$T_1 = 8 \quad Q_1 = \min(6, 3, 5) = 3$$

$$T_2 = 10 \quad Q_2 = \min(0, 2, 4) = 0$$

$$T_3 = 7 \quad Q_3 = \min(2, 4, 3) = 2$$

$$T_4 = 13 \quad Q_4 = \min(0, 0, 0) = 0$$

Lower bound: $\max(11, 10, 9, 13) = 13$.

(2) Schedule 4 first.

$$T_1 = 8 \quad Q_1 = 3$$

$$T_2 = 11 \quad Q_2 = 0$$

$$T_3 = 6 \quad Q_3 = 2$$

$$T_4 = 13 \quad Q_4 = 0$$

Lower bound: $\max(11, 11, 8, 13) = 13$.

(3) Schedule 7 first.

$$T_1 = 8 \quad Q_1 = 3$$

$$T_2 = 11 \quad Q_2 = 0$$

$$T_3 = 6 \quad Q_3 = 2$$

$$T_4 = 14 \quad Q_4 = 0$$

Lower bound: $\max(11, 11, 8, 14) = 14$.

So this gives no increase here, mainly due to the small processing times following M_3 .

Now we use the last method to calculate one new bound.

Schedule 1 first. Then we have:

$$\text{on } M_1 : t_1 = 0 \quad p_1 = 3 \quad q_1 = 6$$

$$t_4 = 3 \quad p_4 = 2 \quad q_4 = 3$$

$$t_7 = 3 \quad p_7 = 3 \quad q_7 = 5$$

$$\text{on } M_2 : t_2 = 3 \quad p_2 = 2 \quad q_2 = 4$$

$$t_9 = 7 \quad p_9 = 2 \quad q_9 = 2$$

$$t_{12} = 4 \quad p_{12} = 3 \quad q_{12} = 0$$

$$\text{on } M_3 : t_5 = 5 \quad p_5 = 1 \quad q_5 = 2$$

$$t_8 = 6 \quad p_8 = 1 \quad q_8 = 4$$

$$t_{11} = 0 \quad p_{11} = 4 \quad q_{11} = 3$$

$$\text{on } M_4 : t_3 = 5 \quad p_3 = 4 \quad q_3 = 0$$

$$t_6 = 6 \quad p_6 = 2 \quad q_6 = 0$$

$$t_{10} = 9 \quad p_{10} = 2 \quad q_{10} = 0$$

So now we have to solve these four one machine problems.

On M_1 : initial solution: 1 : 0 - 3 - 9 *)

7 : 3 - 6 - 11

4 : 6 - 8 - 11

*) Starting at 0, processed at 3, finished at 9.

This solution is optimal by the remark preceding lemma 5.4.2.A.; $C_{\max} = 11$.

On M_2 : initial solution: 2 : 3 - 5 - 9
 12 : 5 - 7 - 7
 9 : 7 - 9 - 11

The same remark does not apply.

$LB_1 = \max(9, 11, 7) = 11$; $LB_2 = 3 + 7 + 0 = 10$.

So $LB = \max(11, 10) = 11$ and the solution is optimal, being equal to the lower bound; $C_{\max} = 11$.

On M_3 : initial solution: 11 : 0 - 4 - 7
 5 : 5 - 6 - 8
 8 : 6 - 7 - 11

LB being 11, this solution is again optimal; $C_{\max} = 11$.

On M_4 : initial solution: 3 : 5 - 9 - 9
 6 : 9 - 11 - 11
 10 : 11 - 13 - 13

Optimal by the same remark as on M_1 ; $C_{\max} = 13$.

So the bound on this branch is not further increased and remains 13.

The reader may well wonder if this complicated method ever leads to significantly better solutions. There is, however, convincing evidence for this. Attacking some old problems with this algorithm, Bratley, Florian and Robillard found an initial solution for one of them that was better than the best previously known one; the finally best solution they found was significantly better. We must strike a somber note nevertheless, because optimality has not been proved for the two problems mentioned above (resp. 5/20 and 10/10 ones), leading the authors to express their belief that "methods other than bounds must be used to further curtail the tree search".

So we see the best here is by far not good enough yet, and this counts even stronger for the implicit enumeration algorithm by Balas [8].

Balas' algorithm boils down to resolve the disjunctive arcs heuristically and then gradually improve the so found feasible solution by reversing one disjunctive arc at the time. It is easy to see that the only way to decrease the length of the critical path is by reversing those arcs that are in the present critical path C . At any stage we calculate the effect of reversing any disjunctive arc in C , reverse the one that gives the greatest effect and fix the reversed arc temporarily. Thus at any stage we have a fixed set of arcs F ; the longest path in the graph formed by \mathcal{N} and $\mathcal{C} \cup F$ is obviously a lower bound and we can backtrack if the lower bound surpasses the present best solution.

We do not pay any more attention to this algorithm, because it is computationally very much inferior to the algorithm of Bratley, Florian and Robillard treated above. Repeating the final remark of the latter authors, we can only stress that, despite recent advances, present branch-and-bound methods are not likely to solve satisfactorily the $n|m|G|F_{\max}$ scheduling problem^{*)}.

*) We would like to point out here an interesting link between resource-constrained project scheduling and the $n|m|G|F_{\max}$ problem. In the former problem we can also use the disjunctive graph model; we only have to check then if our (partial) solution is resource-feasible. For details, see Balas [10], Gorenstein [42] and also Schrage [107] for a slightly different approach.

6. Scheduling in economic reality

6.1. Present situation

A very regrettable aspect of this final chapter is that it is going to be too short. Operations research is a section of mathematics where researchers are typically concerned about the applicability of their work. Many an article has appeared where the main accent is on the development of a mathematical model that can be subjected to existing mathematical techniques, instead of on the development of a technique itself. In view of the fact that one feels that decisions regarding an optimal sequence of activities are certainly not rare ones, the lack of "case studies" in scheduling is downright disappointing. As far ago as 1961 Sisson [92] wrote: "I have "heard" of several actual applications of sequencing theory to several actual cases during the past year, but no results have been announced. (...) It is hoped that the use of sequencing theory in an operating situation will be described soon ...". Nonetheless, the situation has not changed much in the meanwhile. A small number of heuristically solved problems has been reported (e.g. Burstall [20]), but Sisson's wish has hardly been fulfilled. This curious phenomenon deserves some more attention.

We think there are three sides to the explanation of the apparent lack of applicability of machine scheduling theory.

In the first place we can again quote Pounds [80]: "The job-shop scheduling problem is not recognized by most factory schedulers, because for them, in most cases, no scheduling problem exists. That is, there is no scheduling problem for them, because the organization which surrounds the schedulers reacts to protect them from strongly interdependent sequencing decisions (...). Computationally difficult scheduling problems do not arise,

because those constraints that would create them, are removed when they become active". If Pounds is correct here, a great deal has been explained already. In fact, it is fairly plausible that pressure on organizations to work with optimal schedules is fairly low, that due-dates are set with a wide safety margin and that all kinds of mechanisms exist that can cope with the unpleasant consequences of a bad schedule. Even so, one can still hope that, once a theoretically derived schedule is carefully and successfully implemented, management will become more aware of the possibilities that lie ahead. Or will they?

In trying to answer that question, we arrive at the second aspect that we want to mention here. Suppose a real-life machine scheduling problem has been isolated and can be solved purely by theory. Will existing theory be of any help? There are several reasons to at least doubt this, and one of them can be found in 2.2., where all the restrictive assumptions are mentioned that we often find in scheduling theory. One does not need a great deal of business experience to see that most of these assumptions are patently unrealistic. To mention but a few criticisms: in general jobs will not be available at the same time, nor will they be of equal importance (all customers are equal, but some are probably more equal than others). Also, in general jobs will just have to be ready on a fixed date; and machines too are likely not to be continuously available, since, for instance, they may very well break down. Technologically speaking, it is highly unrealistic that each job passes all machines, and each machine only once; equally unrealistic is the assumption that lap-phasing, assembly or job-splitting cannot occur. And perhaps the most improbable assumption of all is the determinate nature of the problem: in economic reality there are always risks and uncertainties that will spoil the beautiful theory.

Put like this, things look very bleak indeed. Are all these objections, valid as they may be, really that serious? In general, we are rather optimistic on this point. Several of the objections can be incorporated in the model: we can attach weights to the jobs that indicate their relative importance, we can even assume there are precedence constraints among them, we can set due-dates d_k and introduce release-dates r_k . The disjunctive graph model is more flexible than we have presented it; it can easily incorporate assembly operations and jobs that only pass a subset of the machines or pass a machine twice or more. Also we have seen that job splitting sometimes even simplifies the solution. With regards to variation in the processing times, Conway, Maxwell and Miller [24] report that optimal solutions of machine scheduling problems are fairly insensitive to changes in $p_{k\ell}$. Most important: when a situation is theoretically under control, sudden emergencies such as high priority jobs or breakdowns need not worry us too much.

Now it cannot be denied that for a certain type of organisation the assumption about a fixed set of jobs is possibly too unrealistic; jobs arrive continuously and our static theory can indeed be of little use. Fortunately, however, we can refer here to a growing theory on queues, waiting lines, etc., while concluding at the same time that the deterministic theory will be mainly applicable to often recurring routine processes. Nevertheless, artificial though the model may be, we do not think that it can only serve as an object of mathematical "Spielerei". Falling back on our first point, practical experience will determine if the model has to be adapted so strongly that present theory would be worthless; again, we are fairly optimistic about the outcome. It remains disturbing all the same that of known applications most have been of the heuristic kind. Perhaps a partial explanation of this can be found in our third aspect.

At first sight, this third aspect of the present theory will seriously hinder application: we refer to the unrealistic optimality criteria. Not unreasonable, Sisson [92] points out that "the ultimate desire is to optimize the objectives of a larger organization (e.g., profits). This requires knowing how the specific situation relates to the whole, knowledge which we do not have. Thus, for research purposes, one optimizes a lesser criterion chosen in some reasonable way". We know the choice most researchers have made: F_{\max} is used by far the most frequently, followed at a respectable distance by \bar{F} , $\sum \alpha_k F_k$, L_{\max} , \bar{T} and $\sum \alpha_k T_k$.

Now, knowing what to produce, the obvious optimality criterion is to minimize total opportunity cost, i.e. those (controllable) costs that reflect our loss with regards to an ideal situation. Deriving an expression for opportunity costs, Gupta [45] has compared the performance of several criteria with regards to this new one. He arrived at the disturbing result that in fact F_{\max} did worst of all and was only very rarely in accordance with opportunity costs. However, at this point as well we are slightly more optimistic. We shall also derive an expression for the opportunity costs and indicate the relation with our present set of criteria (which Gupta does not do). What are the sequence-dependent components of opportunity cost?

(1) Operation costs

We only have to include those costs if we have sequence-dependent set-up times c_{jkl} when J_j precedes J_k at M_ℓ . If not, total operations cost will simply be equal to

$$\sum_{\ell} m_{\ell} \left(\sum_k p_{k\ell} \right),$$

where m_{ℓ} is the machine cost per time unit. This obviously is a sequence-independent constant.

(2) In-process inventory costs

These costs are caused by the fact that during the production process, semi-finished jobs are waiting in the shop, representing tied up capital that could have been used profitably elsewhere.

If return on investment is r , the raw material value of J_k is b_k , the sequence for J_k is $\{M_{k_1}, \dots, M_{k_m}\}$ and machine M_ℓ adds v_ℓ to the value of the product, then the total capital tied up in J_k during the production process is:

$$r[b_k W_{kk_1} + (b_k + v_{k_1}) W_{kk_2} + \dots + (b_k + v_{k_1} + \dots + v_{k_m}) W_{kk_m}] =$$

$$\sum_{\ell} c_{k\ell} W_{k\ell}$$

$$\text{where } c_{k\ell} = r(b_k + \sum_{\substack{(k,j) \\ < (k,\ell)}} v_j).$$

Suppose now, that we can find a reasonably average value β_k , so that costs with regards to J_k are equal to

$$\sum_{\ell} r\beta_k W_{k\ell} = r\beta_k \sum_{\ell} W_{k\ell}$$

So total costs are

$$r \sum_k \beta_k (\sum_{\ell} W_{k\ell}) \quad (1)$$

(3) Penalty costs for late deliveries

Often due-dates d_k will have been set, and if jobs are not completed by then, penalty costs are incurred. These may be of an administrative nature, they may be due to penalty clauses in the contract or simply due to loss of goodwill.

The last factor induces us to assume a positive effect if jobs are completed ahead of their due-date; something that may well

be appreciated by the customer. If we estimate the cost per time unit after d_k as e_k and the positive effect per time unit before d_k as f_k , the total cost will be

$$\sum_k (e_k T_k - f_k E_k).$$

If e_k and f_k do not differ too much, we can both replace them by ϵ_k , and get

$$\sum_k \epsilon_k (T_k - E_k) = \sum_k \epsilon_k L_k \quad (2)$$

(4) Machine idle costs

Obviously, machines standing idle cause a loss to the organization, since they could have performed other useful work during that period. If $I_{k\ell}$ is the time M_ℓ has to wait for J_k , $I_{n+1,\ell}$ represents the time between the finish of last job on M_ℓ and the completion date of all jobs and ρ_ℓ represents the net loss on M_ℓ per time-unit, we have for total costs

$$\sum_\ell \rho_\ell (\sum_k I_{k\ell}) \quad (3)$$

where the last summation is taken over $k = 1, \dots, n+1$.

Taking (1), (2) and (3) together (and therefore assuming sequence-independent set-up costs) we get for total opportunity cost OC:

$$OC = r \sum_k \beta_k (\sum_\ell W_{k\ell}) + \sum_k \epsilon_k L_k + \sum_\ell \rho_\ell (\sum_k I_{k\ell})$$

where r , β_k , ϵ_k and ρ_ℓ are known constants.

Now, we have:

$$\sum_{\ell} W_{k\ell} = F_k - \sum_{\ell} P_{k\ell}$$

$$L_k = F_k - d_k$$

$$\sum_k I_{k\ell} = F_{\max} - \sum_k P_{k\ell}$$

so:

$$OC = \sum (r \beta_k + \epsilon_k) F_k + (\sum_{\ell} \rho_{\ell}) F_{\max} - \sum_{k,\ell} (r \beta_k - \rho_{\ell}) P_{k\ell} - \sum_k \epsilon_k d_k,$$

so, disregarding the last two (sequence-independent) constants, and putting $\alpha_k^* = r \beta_k + \epsilon_k$, $\alpha^* = \sum_{\ell} \rho_{\ell}$, we would have to minimize

$$\sum \alpha_k^* F_k + \alpha^* F_{\max}.$$

Obviously, this criterion appears nowhere in theory. However, with existing methods we can probably get a reasonable approximation by first solving according to F_{\max} ; next we solve according to

$$\alpha_1^* F_1 + \dots + \alpha_{k-1}^* F_{k-1} + (\alpha_k^* + \alpha^*) F_k + \alpha_{k+1}^* F_{k+1} + \dots + \alpha_n^* F_n,$$

where J_k is the job that finished last in the F_{\max} optimal schedule.

So even here things are not as bleak as they looked. What, then, can we predict about the future of scheduling?

6.2. Future developments

First, and most obviously, there remains a lot of theoretical work to be done. Gaps in existing theory have been pointed out

frequently in this report; there is no need to repeat them here. If we look at progress made already, we may expect interesting new developments during the coming years. For - and this is a second point - mathematical interest in the scheduling problem seems to be growing; many articles appear, many researchers are interested, because basically scheduling problems are intriguing, challenging and fun to work at.

There is a dangerous side to all this mathematical activity: as happened in game theory, reality may move further and further away. So one can only hopefully repeat Sisson's wish for applications to be made and reported. Surely one of the many areas, where the machine scheduling model seems appropriate, can provide a good start?

We would nevertheless not be surprised if, for the years to come, good heuristic methods remain of the utmost importance. However, in the long run, nothing is as practical as a good theory. If this report can contribute at all to inspire new practical-theoretical work, it has more than served its purpose.

Bibliography

- [1] T.S. Arthanari & A.C. Mukhopadhyay, "A note on a paper by W. Szwarc", NRLQ 18 (1971), 135-138; $n|2|F|F_{\max}|CA$.
- [2] S. Ashour, "Sequencing theory", Springer Verlag (1972).
- [3] S. Ashour & R.G. Parker, "A precedence graph algorithm for the shop scheduling problem", ORQ 22 (1971), 165-175; $n|m|G|F_{\max}|H,A$.
- [4] P.C. Bagga & N.K. Chakravarti, "Optimal m-stage production schedule", J. Can. Op. Res. Soc. 6 (1968), 71-78; $n|m|P|F_{\max}$.
- [5] K.R. Baker, "Procedures for sequencing tasks with one resource type", Int. J. Prod. Res. 11 (1973), 125-138; $n|1||F_{\max}, \Sigma \alpha_k F_k | CA$.
- [6] M.S. Bakshi & S.R. Arora, "The sequencing problem", Mngmt. Sc. 16 (1969), B-247-263; $n|m|G|F_{\max}|CA,H$.
- [7] E. Balas, "Discrete programming by the filter method", OR 15 (1967), 915-957; $n|m|G|F_{\max}|IP$.
- [8] E. Balas, "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm", OR 17 (1969), 941-957; $n|m|G|F_{\max}|BB$.
- [9] E. Balas, "Machine sequencing: disjunctive graphs and degree-constrained subgraphs", NRLQ 17 (1970), 1-10; $n|m|G|F_{\max}|IP$.

- [10] E. Balas, "Project scheduling with resource constraints", in: "Applications of mathematical programming techniques", ed. by E.M.L. Beale, The English Universities Press (1970).
- [11] M.L. Balinski, "Integer programming: methods, uses, computation", Mngmt. Sc. 12 (1965), 253-313.
- [12] E.M.L. Beale, "Survey of integer programming", ORQ 16 (1965), 219-228.
- [13] M.J. Beckmann, "Dynamic programming of economic decisions", Springer Verlag (1968).
- [14] M. Bellmore & G.L. Nemhauser, "The travelling salesman problem: a survey", OR 16 (1968), 538-559.
- [15] A.J.M. Beulens & A.A. van den Hark, "Oplosmethoden voor deterministische volgorde problemen", T.H.Eindhoven (1972).
- [16] E.H. Bowman, "The schedule-sequencing problem", OR 7 (1959), 621-624; $n|m|G|?|IP$.
- [17] P. Bratley, M. Florian & P. Robillard, "Scheduling with earliest start and due-date constraints", NRLQ 18 (1971), 511-519; $n|1||F_{\max}, T_{\max} = 0|IP, BB$.
- [18] G.H. Brooks & C.R. White, "An algorithm for finding optimal or near optimal solutions to the production scheduling problem", J. Ind. Eng. 16 (1965), 34-40; $n|m|P|F_{\max}|BB$.
- [19] A.P.G. Brown & Z.A. Lomnicki, "Some applications of the "branch-and-bound" algorithm to the machine scheduling problem", ORQ 17 (1966), 173-186; $n|m|P|F_{\max}|BB$.

- [20] R.M. Burstall, "A heuristic method for a job-scheduling problem", ORQ 17 (1966), 291-304; $n|3||H$.
- [21] J.M. Charlton & C.C. Death, "A method of solution for general machine scheduling problems", OR 18 (1970), 689-707; $n|m|G|F_{\max}|BB$.
- [22] J.M. Charlton & C.C. Death, "A generalized machine scheduling algorithm", ORQ 21 (1971), 127-134; $n|m|G|F_{\max}|BB$.
- [23] E. Coffman & R. Muntz, "Optimal preemptive scheduling on two-processor systems", I.E.E.E. Trans. Comput. 18, 1014; $n|m_p|(J6)|F_{\max}|CA$.
- [24] R.W. Conway, W.L. Maxwell & L.W. Miller, "Theory of scheduling", Addison-Wesley (1967).
- [25] J. Day & M.P. Hottenstein, "Review of sequencing research", NRLQ 17 (1970), 11-39.
- [26] L.C. Driscoll & L. Suyemoto, "Heuristics for resolution of logical scheduling conflicts" in: "Proceedings of the fourth international conference on operational research", Wiley-Interscience (1966).
- [27] R.A. Dudek & O.F. Teuton, Jr., "Development of M-stage decision rules for scheduling N jobs through M machines", OR 12 (1964), 471-497; $n|m|P|F_{\max}|H$.
- [28] J.M. Dutton, "Production-scheduling - a behavioural model", Int. J. Prod. Res. 3 (1964); H.

- [29] S.E. Elmaghraby, "The machine scheduling problem - review and extensions", *NRLQ* 15 (1968), 205-232.
- [30] H. Emmons, "One-machine sequencing to minimize certain functions of job tardiness", *OR* 17 (1969), 701-715; $n|1|\bar{T}|CA$.
- [31] M.L. Fisher, "Optimal solution of scheduling problems using Lagrange multipliers (revised)", report 7210, University of Chicago.
- [32] M. Florian, P. Trépan & G. McMahon, "An implicit enumeration algorithm for the machine sequencing problem", *Mngmt. Sc.* 17 (1971), B-782-792; $n|m|G|F_{\max}|BB$.
- [33] M. Florian & N. Sang, "A note on lower bounds for the machine scheduling problem", publ. # 49, Département d'Informatique, Université de Montréal; $n|m|G|F_{\max}|BB$.
- [34] M. Florian, P. Bratley & P. Robillard, "On sequencing with earliest starts and due-dates with application to computing bounds for the $(n|m|G|F_{\max})$ problem", *NRLQ* 20 (1973), 57-67; $n|m|G|F_{\max}|BB$.
- [35] A.M. Geoffrion & R.E. Marsten, "Integer programming: a framework and state-of-the-art survey", *Mngmt. Sc.* 18 (1972), 465-491.
- [36] W.S. Gere, Jr., "Heuristics in job shop scheduling", *Mngmt. Sc.* 13 (1966), 167-190; H.
- [37] B. Giffler, "Schedule algebras and their use in formulating general systems simulations" in: "Industrial scheduling", ed. by J.F. Muth and G.L. Thompson, Prentice-Hall (1963); $n|m||F_{\max}|A$.

- [38] B. Giffler, "Mathematical solution of production planning and scheduling problems", IBM ASDD Technical Report (1960); $n|m|F_{\max}|A$.
- [39] B. Giffler & G.L. Thompson, "Algorithms for solving production-scheduling problems", OR 8 (1960), 487-503; $n|m|G|F_{\max}|CA$.
- [40] P.C. Gilmore & R.E. Gomory, "Sequencing a one state-variable machine: a solvable case of the travelling salesman problem", OR 12 (1964), 655-679.
- [41] C.R. Glassey, "Minimum change-over scheduling of several products on one machine", OR 16 (1968), 342-352; $n|1| \# \text{ change-overs}|DP,CA$.
- [42] S. Gorenstein, "An algorithm for project (job) sequencing with resource constraints", OR 20 (1972), 835-850; $n|m|G,(J2)|F_{\max}|BB$.
- [43] H.H. Greenberg, "A branch-bound solution to the general scheduling problem", OR 16 (1968), 353-361; $n|m|G|F_{\max}|BB$.
- [44] J.N.D. Gupta, "A functional heuristic algorithm for the flow shop scheduling problem", ORQ 22 (1971); $n|m|F|F_{\max}|H$.
- [45] J.N.D. Gupta, "Economic aspects of production scheduling systems", J. Op. Res. Soc. of Japan 13 (1971), 169-193.
- [46] J.N.D. Gupta, "Optimal flow shop scheduling with due-dates and penalty costs", J. Op. Res. Soc. of Japan 14 (1971), 35-46; $n|m|F|?|BB$.
- [47] J.N.D. Gupta, "The generalized n-job, m-machine scheduling problem", Opsearch 8 (1971), 173-185; $n|m|G|F_{\max}|IP$.

- [48] C. Haehling Von Lanzenuer & R.C. Himes, "A linear programming solution to the general sequencing problem", J. Can. Op. Res. Soc. 18 (1970), 129-134; $n|m|G|F_{\max}|LP$.
- [49] J. Heller, "Some numerical experiments for an $M \times J$ flow shop and its decision theoretical aspects", OR 8 (1960), 178-184; H.
- [50] W.A. Horn, "Single-machine job sequencing with treelike precedence ordering and linear delay penalties", SIAM J. Appl. Math. 23 (1972), 189-202; $n|1|(J6)|\sum_k \alpha_k T_k|CA$.
- [51] T.C. Hu, "Parallel sequencing and assembly line problems", OR 9 (1961), 841-848; $n|m_p|(J6)|F_{\max}|CA$.
- [52] E. Ignall & L. Schrage, "Application of the branch-and-bound technique to some flow shop scheduling problems", OR 13 (1965), 400-412; $n|2|F|\bar{F}|BB$, $n|3|F|F_{\max}|BB$.
- [53] J.R. Jackson, "An extension of Johnson's results on job lot scheduling", NRLQ 3 (1956), 201-203; $n|2|G|F_{\max}|CA$.
- [54] S.M. Johnson, "Optimal two- and three-stage production schedules with set-up times included", NRLQ 1 (1957), 61-68; $n|2|F|F_{\max}|CA$, $n|3|F|F_{\max}|CA$.
- [55] S.M. Johnson, "Discussion: sequencing n jobs on two machines with arbitrary time lags", Mngmt. Sc. 5 (1958), 299-303; $n|2|F|F_{\max}|CA$.
- [56] W. Karush, "A counter-example to a proposed algorithm for optimal sequencing of jobs", OR 13 (1965), 323-325; $n|m|P|F_{\max}$.

- [57] E.L. Lawler, "On scheduling problems with deferral costs", Mngmt. Sc. 11 (1964), 280-288; $n|m_p|\Sigma c_k(t)|DP,LP$.
- [58] E.L. Lawler & D.E. Wood, "Branch-and-bound methods: a survey", OR 14 (1966), 699-718.
- [59] E.L. Lawler & J.M. Moore, "A functional equation and its application to resource allocation and sequencing problems", Mngmt. Sc. 16 (1969), 77-84; $n|1|\bar{T},\Sigma\alpha_k E_k|DP$.
- [60] E.L. Lawler, "Optimal sequencing of a single machine subject to precedence constraints", Mngmt. Sc. 19 (1973), 544-546; $n|1|(J6)|c_k(t)_{max}|CA$.
- [61] F.K. Levy, G.L. Thompson & J.D. Wiest, "Mathematical basis of the critical path method" in: "Industrial scheduling", ed. by J.F. Muth and G.L. Thompson, Prentice-Hall (1963).
- [62] J. Little, K. Murty, D. Sweeney & C. Karel, "An algorithm for the travelling salesman problem", OR 11 (1963), 972-989.
- [63] Z.A. Lomnicki, "A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem", ORQ 16 (1965), 89-100; $n|3|F|F_{max}|BB$.
- [64] A.S. Manne, "On the job shop scheduling problem", OR 8 (1960), 219-223; $n|1|(J6)|F_{max}|IP$.
- [65] R. McNaughton, "Scheduling with deadlines and loss functions", Mngmt. Sc. 6 (1959), 1-12; $n|1|\Sigma\alpha_k F_k|CA$, $n|m_p|\bar{F},\Sigma\alpha_k F_k|CA$.
- [66] P. Mellor, "A review of job shop scheduling", ORQ 17 (1966), 161-171.

- [67] G. Mensch, "The inspection problem", Math. Operationsforsch. und Statist. 2 (1971), 261-269 (generalized $n|m|G$ problem).
- [68] L.G. Mitten, "Sequencing n jobs on two machines with arbitrary time lags", M. Sc. 5 (1958), 293-298; $n|2|F|F_{\max}|CA$.
- [69] J.M. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs", Mngmt. Sc. 14 (1968), 102-109; $n|1||\# \text{ tardy jobs}|CA$.
- [70] H. Müller-Merbach, "Optimale Reihenfolgen", Springer Verlag (1970).
- [71] J.F. Muth & G.L. Thompson (eds.), "Industrial scheduling", Prentice-Hall (1963).
- [72] I. Nabeshima, "The order of n items produced on m machines", J. Op. Res. Soc. of Japan 3 (1961), 170-175: $n|m|P|F_{\max}|CA$.
- [73] I. Nabeshima, "The order of n items produced on m machines (II)", J. Op. Res. Soc. of Japan 4 (1961), 1-8; $n|m|P|F_{\max}|CA$.
- [74] I. Nabeshima, "Sequencing on two machines with start lag and stop lag", J. Op. Res. Soc. of Japan 5 (1963), 97-101; $n|2|F|F_{\max}|CA$.
- [75] I. Nabeshima, "Computational solution to the m -machine scheduling problem", J. Op. Res. Soc. of Japan 7 (1965), 93-103; $n|m|P|F_{\max}|CA,DP$.
- [76] I. Nabeshima, "General scheduling algorithms with applications to parallel scheduling and multiprogramming scheduling", J. Op. Res. Soc. of Japan 14 (1971), 72-99; $n|m|G|F_{\max}|BB$.

- [77] L. Németi, "Das Reihenfolgeproblem in der Fertigungsprogrammierung und Linearplanning mit logischen Bedingungen", *Mathematica* 6 (1964), 87-99; $n|m|G|F_{\max}|BB$.
- [78] P. Nepomiastchy, "Application of the penalty technique to solve a scheduling problem and comparison with combinatorial methods", *Rapport de Recherche no. 7*, Institut de Recherche d'Informatique et d'Automatique (1973); $n|m|G|F_{\max}|NLP$.
- [79] T.A.J. Nicholson, "A method for optimizing permutation problems and its industrial applications" in: "Combinatorial mathematics and its applications", ed. by D.J.A. Welsh, Academic Press (1971).
- [80] W.F. Pounds, "The scheduling environment" in: "Industrial scheduling", ed. by J.F. Muth and G.L. Thompson, Prentice-Hall (1963).
- [81] J.F. Raimond, "Minimaximal paths in disjunctive graphs by direct search", *IBM J. of Res. and Dev.* 13 (1969), 391-399; $n|m|G|F_{\max}|IP$.
- [82] S.S. Reddi & C.V. Ramamoorthy, "On the flow shop sequencing problem with no wait in process", *ORQ* 23 (1972), 323-331; $n|m|F, (J4)|F_{\max}|CA$.
- [83] J.F. Rial, "The resolution of conditional scheduling conflicts" in: "Proceedings of the fourth international conference on operational research", Wiley-Interscience (1966).
- [84] J.G. Root, "Scheduling with deadlines and loss functions on k parallel machines", *Mngmt. Sc.* 11 (1965), 460-475; $n|m_p|\bar{T}|CA$.

- [85] M.H. Rothkopf, "Scheduling independent tasks on parallel processors", Mngmt. Sc. 12 (1966), 437-447;
 $n|m_p||\Sigma\alpha_k F_k|DP$.
- [86] B. Roy & B. Sussmann, "Les problèmes d'ordonnancement avec contraintes disjonctives", note DS no. 9 bis SEMA (1964).
- [87] B. Roy, "Problems and methods with multiple objective functions", Math. Progr. 1 (1971), 239-267.
- [88] V.K. Sahney, "Single-server, two-machine sequencing with switching time", OR 20 (1972), 24-36; $n|2|(M3)|F_{max}|CA$.
- [89] L. Schrage, "Solving resource-constrained network problems by implicit enumeration - non-preemptive case", OR 18 (1970), 263-278; $n|m|G|F_{max}|BB$.
- [90] L. Schrage, "Solving resource-constrained network problems by implicit enumeration - preemptive case", OR 20 (1972), 668-677; $n|m|G,(JM2)|F_{max}|BB$.
- [91] J. Shwimer, "On the n-job, one-machine, sequence-independent scheduling problem with tardiness penalties: a branch-bound solution", Mngmt. Sc. 18 (1972), B-301-313;
 $n|1||\Sigma\alpha_k T_k|BB$.
- [92] R.L. Sisson, "Sequencing theory" in: "Progress in operations research", ed. by R. Ackoff, Wiley (1961).
- [93] W.E. Smith, "Various optimizers for single-stage production", NRLQ 3 (1956), 59-66; $n|1||\Sigma\alpha_k F_k, T_{max}|CA$.
- [94] R.D. Smith & R.A. Dudek, "A general algorithm for solution of the n-job, M-machine sequencing problem of the flow shop", OR 15 (1967), 71-82; $n|m|P|F_{max}|CA$.

- [95] R.D. Smith & R.D. Dudek, "Errata", OR 17 (1969), 756;
 $n|m|P|F_{\max}|CA$.
- [96] A.E. Story & H.M. Wagner, "Computational experience with integer programming for job shop scheduling" in: "Industrial scheduling", ed. by J.F. Muth and G.L. Thompson, Prentice-Hall (1963); $n|3|F|F_{\max}|IP$.
- [97] L.B.J.M. Sturm, "A simple optimality proof of Moore's sequencing algorithm", Mngmt. Sc. 17 (1970), 116-118;
 $n|1||\# \text{ tardy jobs}|CA$.
- [98] B.G. Sussmann, "Scheduling problems with interval disjunctions", Zeitschrift für O.R. 16 (1972), 165-178;
 $n|m|G|F_{\max}|BB$.
- [99] W. Szwarc, "On some sequencing problems", NRLQ 15 (1968), 127-155; $n|2|F,G|F_{\max}|CA$, $n|3|F|F_{\max}|CA$, $2|m|G|F_{\max}|CA$,
 $n|m|G|F_{\max}|H$.
- [100] W. Szwarc, "Elimination methods in the $m \times n$ sequencing problem", NRLQ 18 (1971), 295-305; $n|m|P|F_{\max}|CA$.
- [101] J. Terno, "Algorithmen für das klassische Maschinenbelegungsproblem", Math. Operationsforsch. und Statist. 3 (1972), 195-201; $n|m|P|F_{\max}|CA$.
- [102] H.M. Wagner, "An integer programming model for machine scheduling", NRLQ 6 (1959), 131-140; $n|m|F|F_{\max}|IP$.
- [103] D.A. Wismer, "Solution to the flow shop scheduling problem with no intermediate queues", OR 20 (1972), 689-697;
 $n|m|F,(J4)|F_{\max}|CA$.

Added later:

- [104] P. Bratley, M. Florian & P. Robillard, "Scheduling with earliest start and due-date constraints on multiple machines", report 111, Département d'Informatique, Université de Montréal (1973); $n|m_p||F_{\max}, T_{\max} = 0|BB$.
- [105] W. Gapp, P.S. Mankekar & L.G. Mitten, "Sequencing theory operations to minimize in-process inventory costs", Mngmt. Sc. 11 (1965), 476-484; $n|1|(J6)|\Sigma\alpha_k F_k$.
- [106] L. Schrage, "A bound based on the equivalence of min-max-completion time and min-max-lateness scheduling objectives", report 7042, University of Chicago; $n|m|G|F_{\max}|BB$.
- [107] L. Schrage, "Obtaining optimal solutions to resource constrained network scheduling problems", (1971); $n|m|G|F_{\max}, \Sigma\alpha_k L_k, \Sigma\alpha_k T_k|BB$.

