TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Machine scheduling and Lagrangian relaxation

Citation for published version (APA):
Velde, van de, S. L. (1991). *Machine scheduling and Lagrangian relaxation*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Centrum voor Wiskunde en Informatica. https://doi.org/10.6100/IR350591

DOI:
10.6100/IR350591

Document status and date:
Published: 01/01/1991

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

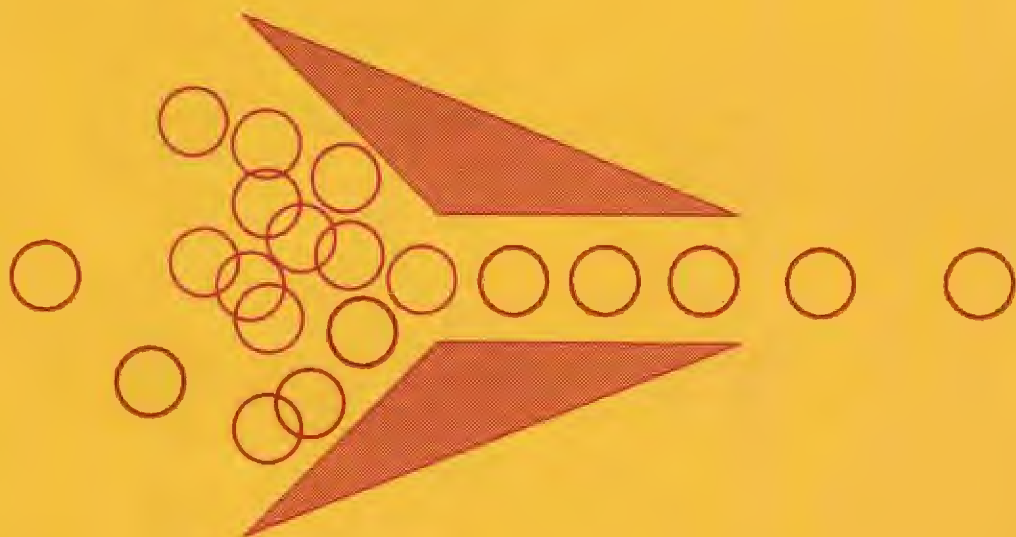
Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 16. Nov. 2023

MACHINE SCHEDULING AND LAGRANGIAN RELAXATION



Steef van de Velde

MACHINE SCHEDULING
AND
LAGRANGIAN RELAXATION

MACHINE SCHEDULING
AND
LAGRANGIAN RELAXATION

PROEFSCHRIFT
ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof. dr. J.H. van Lint, voor
een commissie aangewezen door het College van
Dekanen in het openbaar te verdedigen op
vrijdag 5 april 1991 te 16.00 uur
door

STEVEN LEENDERT VAN DE VELDE

geboren te Sint Philipsland

1991
CWI, Amsterdam

Dit proefschrift is goedgekeurd door
de promotor

Prof. dr. J. K. Lenstra

Voor mijn vader

Acknowledgements

Many people contributed in one way or another to this thesis. Gerard Kindervater, Ben Lageweg, Martin Savelsbergh, and Peter de Waal were always prepared to help me with text processing and computer programming. Tobias Baanders designed the cover. Emile Aarts, Mohamed Dessouky, Antoon Kolen, dr. Lebedev, Anne Marie van Luijt, Henk Oosterhout, Udatta Palekar, Chris Potts, Maurice Queyranne, and Auke Woerlee gave valuable comments on the thesis or on the papers constituting it. Bert Gerards helped me with two particular proofs. Han Hoogeveen conducted joint research on the subjects of the last two chapters; he gave candid comments on the others. Laurence Wolsey reviewed an earlier version scrupulously; his comments on Lagrangian relaxation were particularly helpful.

It has been a great privilege and pleasure to have Jan Karel Lenstra as a supervisor. I have benefited and learned a lot from his expertise.

I am grateful to them all.

Steef van de Velde

TABLE OF CONTENTS

1. Introduction 1
1.1. Machine scheduling 1
1.2. Combinatorial optimization 8
1.3. Lagrangian relaxation and duality 17
1.4. Machine scheduling and Lagrangian relaxation 35

2. Single-machine scheduling 39
2.1. The Lagrangian dual of $1|prec|\Sigma w_j C_j$ 40
2.2. Approximation 47
2.3. Primal decomposition 49
2.4. The total weighted tardiness problem 51

3. Flow-shop scheduling 55
3.1. Introduction 55
3.2. Formulation and relaxation 56
3.3. Dominance criteria 61
3.4. The algorithm 62
3.5. Extensions 64

4. Parallel-machine scheduling 67
4.1. Introduction 67
4.2. Minimizing makespan and its dual problem 69
4.3. Duality-based heuristic search 78
4.4. The branch-and-bound algorithm 80
4.5. Computational experiments 82
4.6. Conclusions 88

5. Common due date scheduling 89
5.1. Introduction 89
5.2. Emmons' matching algorithm for the unrestricted problem 91
5.3. A new lower bound for the restricted variant 91
5.4. A new upper bound for the restricted variant 95
5.5. Branch-and-bound 96
5.6. Computational results 97

6. Just-in-time scheduling 101
6.1. Introduction 102
6.2. The insertion of idle time for a given sequence 104
6.3. The branch-and-bound algorithm 107
6.4. Lower bounds 111
6.5. Computational results 126
6.6. Conclusions 126

References 129

Samenvatting 137

1
Introduction

1.1. Machine scheduling

Motivated and stimulated by the practical relevance of production planning and computer scheduling problems, *scheduling* has become an important area of operations research. In the broadest sense, 'scheduling is the allocation of resources over time to perform a collection of tasks' [Baker, 1974], and the theory of scheduling is concerned with 'the optimal allocation of scarce resources to activities over time' [Lawler, Lenstra, Rinnooy Kan, and Shmoys, 1989] and with 'the optimal utilization of the usually limited resources in accomplishing the variegated tasks or objectives' [Bellman, Esogbue, and Nabeshima, 1982].

We confine ourselves to scheduling problems in which each task or activity requires at most one resource at a time. In this case, scheduling problems are usually seen as problems that concern the scheduling of *jobs* on *machines* of limited capacity and availability. Such problems are traditionally referred to as *machine scheduling* problems. A job consists of an ordered list of operations, each of which requires processing during a certain period of time on some machine. Each machine can process at most one job at a time and is continuously available from time 0 onwards. A job can be processed by at most one machine at a time. A *schedule* specifies for each job when and by which machine it is executed. The objective is to find a schedule that optimizes some criterion function. Usually, this is a function of the job completion times.

The variety of machine environments, job characteristics, and objective functions give rise to a myriad of machine scheduling problems. In this thesis, we consider only *deterministic* machine scheduling problems: we assume perfect knowledge of the data beforehand.

In this introductory chapter, we give a flavor of what machine scheduling problems and their associated solution techniques are about. We familiarize the reader with some scheduling models and concepts in Section 1.1, in which we

consider problems involving the following machine environments: the single-machine shop, the flow shop, and the parallel-machine shop. These machine environments are the subject of further study in the subsequent chapters. In Section 1.2, we point out how machine scheduling problems fit into the broader framework of combinatorial optimization and give an informal introduction to the theory of computational complexity. With the help of this theory, it is possible to classify problems as easy or probably hard to solve.

Introductions to these fields necessarily have to be selective and concise: only those concepts that are relevant for the subsequent chapters are discussed; others are merely touched upon. For more elaborate introductions to the respective areas, we refer to Conway, Maxwell, and Miller [1967], Baker [1974], French [1982], and Lawler, Lenstra, Rinnooy Kan, and Shmoys [1989] for machine scheduling, to Lawler, Lenstra, Rinnooy Kan, and Shmoys [1985] for a guided tour through combinatorial optimization, and to Garey and Johnson [1979] for computational complexity.

The subject of this thesis is the application of Lagrangian relaxation and duality to machine scheduling problems. Although the technique of Lagrangian relaxation is known to be helpful in solving many types of hard combinatorial optimization problems, its use for machine scheduling problems is limited thus far. However, the message of this thesis is that Lagrangian relaxation has much to offer to machine scheduling theory. In Section 1.3, we introduce the basic concepts and issues involved in the application of Lagrangian relaxation, thereby focusing on machine scheduling problems. In Section 1.4, we give an overview of the literature on Lagrangian relaxation applied to machine scheduling problems, describe what our objectives are, and give a preview of the scheduling problems that are dealt with in the subsequent chapters.

1.1.1. *Single-machine scheduling*

The usual setting for the single-machine job shop is as follows. A set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ has to be scheduled on a single machine. Each job J_j $(j = 1, \ldots, n)$ consists of one operation requiring processing during a period of length p_j. Each J_j is only available for processing during a prespecified period: it becomes available at its *release date* r_j and must be completed by its *deadline* $\bar{d}_j$. In addition, each job may have a positive weight w_j, which expresses its importance with respect to the other jobs, and a *due date* d_j, by which it should be completed. The weights and the due dates are typically used to define the objective function. The machine can handle no more than one job at a time and is continuously available from time 0 onwards.

Consider the data of the 5-job example in Table 1.1. All release dates are assumed to be 0, and all deadlines are set to infinity. We have represented an arbitrary schedule in the form of a so-called *Gantt chart* in Figure 1.1. The schedule is feasible in terms of machine capacity and availability: it specifies for each job J_j a completion time C_j such that the jobs do not overlap in their execution, and such that $C_j - p_j \geqslant 0$ for $j = 1, \ldots, n$. Each job once started is processed without interruption. We say that a job is *preempted* if its execution is interrupted and resumed at a later point in time.

	J_1	J_2	J_3	J_4	J_5
p_j	6	3	4	6	8
w_j	5	2	2	2	1

TABLE 1.1. Processing times and weights.

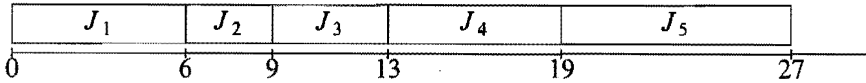


FIGURE 1.1. Gantt chart.

We consider an elementary single-machine problem. Suppose the objective is to find a schedule that minimizes the sum of the weighted completion times, that is, $\Sigma_{j=1}^{n} w_j C_j$. This objective function is often interpreted as a measure for the work-in-process inventory as well as for the speed by which the producer responds to the consumers' demands. The data given in Table 1.1 specify an instance of the problem type of minimizing $\Sigma_{j=1}^{n} w_j C_j$ on a single machine. In general, a problem *instance* is formed by specific choices for the parameters of the problem *type*. We make now the following observations.

OBSERVATION 1.1. In any optimal schedule, the jobs are processed consecutively in the interval $[0, \Sigma_{j=1}^{n} p_j]$.

After all, the objective function is non-decreasing in the job completion times. If there were idle time before the completion of the last job, then the objective value could be reduced by shifting jobs to the left, thereby removing the machine idle time.

OBSERVATION 1.2. There is no optimal schedule in which some job is preempted.

Suppose there were an optimal schedule with some job interrupted and its execution resumed at a later point in time. By processing the different portions of the interrupted job immediately before the execution of its last portion, we do not change its completion time, but reduce the completion times of the jobs that were previously finished between the execution of the first and last portion of the preempted job.

These two observations reduce the single-machine scheduling problem of minimizing $\Sigma_{j=1}^{n} w_j C_j$ to a *sequencing* problem: we should determine the sequence in which the jobs go through the machine. There is a fundamental algorithm by Smith [1956] that solves this problem in an easy way.

THEOREM 1.1. *The single-machine problem of minimizing $\Sigma_{j=1}^{n} w_j C_j$ is solved by processing the jobs in order of non-increasing values w_j / p_j.*

PROOF. First, we prove that such an order is *necessary* for optimality. The proof,

typical of a number of proofs in machine scheduling, proceeds by contradiction and by use of an interchange argument. Suppose there is an optimal sequence, in which J_k is immediately scheduled before J_l although $w_k/p_k < w_l/p_l$. These jobs are hence not processed in compliance with Smith's rule. If C_k is the completion time of J_k, then J_l is completed at time $C_l = C_k + p_l$. Hence, the cost contributed by the two jobs is

$$w_l(C_k + p_l) + w_k C_k. \quad (1.1)$$

If we swap J_l and J_k, the cost of their execution amounts to

$$w_l(C_k + p_l - p_k) + w_k(C_k + p_l). \quad (1.2)$$

Subtracting (1.2) from (1.1) yields

$$w_l p_k - w_k p_l = (w_l/p_l - w_k/p_k)(p_k p_l) > 0,$$

which contradicts the optimality of the first schedule.

Second, we need the observation that each sequence has the same objective value if all jobs have equal ratios w_j/p_j. This is easily established by an interchange argument.

The combination of these two arguments leads to the conclusion that the necessary condition, fully prescribing the scheduling order, is also *sufficient* for optimality. □

Notice that the schedule depicted in Figure 1.1 is optimal for the 5-job problem. In Chapter 2, we consider the same setting except that there are precedence relations between the jobs; this means that each job has a number of jobs, each of which has to precede this job in any feasible sequence. It will turn out that this problem is much more difficult to solve to optimality.

1.1.2. *Flow-shop scheduling*

An m-machine flow shop is described as follows. There are m machines, each of which can handle at most one job at a time and is continuously available from time 0 onwards. There is a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, each of which consists of a chain of m operations. The ith operation of job J_j has to be executed on machine M_i during a positive processing time p_{ij} ($i = 1, \ldots, m$, $j = 1, \ldots, n$). Note that this means that the jobs pass through the machines in the same order. Each job can be executed by at most one machine at a time: operations of the same job may not overlap in their execution.

Consider the 2-machine 5-job example from Baker [1974] in Table 1.2; we have depicted a feasible schedule in Figure 1.2. Observe that both machines process the jobs in the same order.

We address the problem of minimizing the maximum job completion time in the 2-machine flow shop. The maximum completion time $C_{\max} = \max_{1 \leqslant j \leqslant n} C_j$ is referred to as the *makespan*. Note that we have $C_{\max} = 24$ for the schedule in Figure 1.2. In parallel to the single-machine problem, we make some easy observations. First, there is an optimal schedule in which all the operations are performed without any unnecessary delay and without interruption. Second, there is

	J_1	J_2	J_3	J_4	J_5
p_{1j}	1	3	6	7	5
p_{2j}	2	6	6	5	2

TABLE 1.2. Processing times.

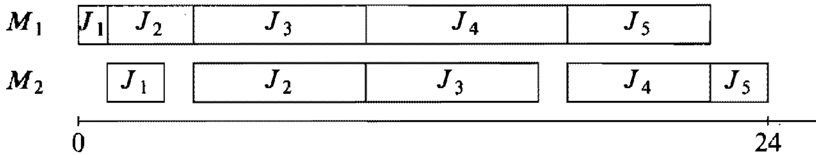


FIGURE 1.2. Gantt chart.

an optimal schedule in which both machines process the jobs in the same order. A schedule with this feature is called a *permutation* schedule. The latter observation, easily validated by an interchange argument, is particularly useful: the scheduling problem reduces again to a sequencing problem. Let σ be some sequence with the jobs reindexed in order of appearance. Using the earlier observations, we note that the minimum makespan for σ can be expressed as

$$C_{\max} = \max_{1 \leqslant k \leqslant n}\left(\sum_{j=1}^{k} p_{1j} + \sum_{j=k}^{n} p_{2j}\right).$$

Observe that the makespan of the schedule in Figure 1.2 can be expressed in this way. We stipulate the following elementary rule, which is due to Johnson [1954].

THEOREM 1.2. *The problem of minimizing the makespan in the 2-machine flow shop is solved by scheduling first the jobs with $p_{1j} \leqslant p_{2j}$ in order of non-decreasing p_{1j}, and then by scheduling the remaining jobs in order of non-increasing p_{2j}.* □

Note that the schedule presented in Figure 1.2 is an optimal schedule for the instance in Table 1.2. For the case of $m \geqslant 3$ machines, there is no easy rule to solve the makespan problem. In Chapter 3, we analyze the 2-machine flow shop with the objective to minimize the sum of the job completion times, that is, $\Sigma_{j=1}^{n} C_j$. It will appear that we have to go through a lot more trouble to solve this problem.

1.1.3. *Parallel-machine scheduling*

Suppose there are m parallel machines available for processing a set of n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each of these machines can handle at most one job at a time. The processing of J_j $(j = 1, \ldots, n)$ on machine M_i $(i = 1, \ldots, m)$ requires a positive uninterrupted period of length p_{ij}. Each job has to be scheduled on exactly one of the m machines. We may assume $n \geqslant m$.

Consider the following 8-job 3-machine example for which the processing times are given in Table 1.3. A feasible schedule is given in Figure 1.3; the length

of the schedule, or the makespan, is equal to 33. This schedule is obtained by simply scheduling each job on the machine that handles it fastest.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
M_1	30	6	3	10	12	11	8	6
M_2	20	10	∞	15	6	6	14	7
M_3	10	11	9	14	14	∞	10	9

TABLE 1.3. Processing time matrix.

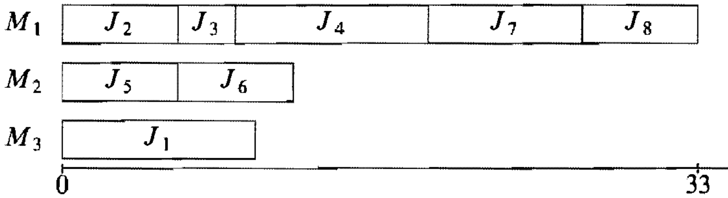


FIGURE 1.3. Gantt chart for the parallel-machine shop.

The objective is to find a schedule of minimum length. Again, we make some easy observations. Apparently, there is an optimal schedule with each machine processing the jobs assigned to it without delay. Furthermore, the order in which each machine processes its jobs is immaterial. This scheduling problem reduces to an *assignment* problem: for a given assignment of jobs to machines, it is easy to construct a corresponding schedule with minimum makespan.

The schedule in Figure 1.3 is not optimal. If we decide to assign J_4 to M_3 rather than to M_1, then we get a schedule with makespan 24, which, as it happens, is not optimal either.

Obvious and intuitively appealing assignment rules cannot be expected to produce optimal solutions for all instances of the problem. We will give evidence for this in Section 1.2. If we insist on finding the optimal solution, then we could follow the approach to enumerate and examine all feasible solutions. Since each of the n jobs can be assigned to m machines, there are m^n such solutions to consider. Since this number grows exponentially with the number of jobs, this approach is viable only for instances of very limited size. An alternative approach is to solve the problem to optimality by means of a *dynamic programming* algorithm. For such an algorithm, we need some *principle of optimality*. If we compare all partial schedules for the jobs $J_1, \ldots, J_j$ that occupy the machines exactly up to times $t_1, \ldots, t_m$, then apparently we need only to consider the one with least cost, as the other schedules, having higher cost, can never lead to an optimal solution. This notion of dominance is valid, since the order in which the jobs are processed on the machines is irrelevant for the length of the schedule.

This optimality principle can be recursively applied in the following way. Let $F_j(t_1, \ldots, t_m)$ denote the minimum cost for scheduling the jobs $J_1, \ldots, J_j$ without idle time subject to the constraint that the last job on M_i is completed at time t_i, for $i = 1, \ldots, m$. The initialization of the recursion is

$$F_0(t_1,\ldots,t_m)=\begin{cases}0, & \text{if } t_i=0 \text{ for } i=1,\ldots,m,\\ \infty, & \text{otherwise},\end{cases}$$

and the recursion for $j=1,\ldots,n$ is given by

$$F_j(t_1,\ldots,t_m)=\min_{1\leqslant i\leqslant m}\max\{t_i,F_{j-1}(t_1,\ldots,t_i-p_{ij},\ldots,t_m)\},$$
$$\text{for } t_i=1,\ldots,T_i,$$

where $T_i=\Sigma_{j=1}^n p_{ij}$. The optimal solution value is then equal to

$$\min_{0\leqslant t_1\leqslant T_1,\ldots,0\leqslant t_m\leqslant T_m} F_n(t_1,\ldots,t_m),$$

and the optimal schedule can be found by backtracing. The recursion shows that the *time* required to execute this algorithm grows linearly with the number of jobs and the processing times, and exponentially with the number of machines. Since we may assume that $n\geqslant m$, the dynamic programming algorithm seems to be preferable to complete enumeration, as the time for the latter depends exponentially on n. However, the linear dependency on the processing times may be prohibitive, even in case of few machines. In addition, the *space* required to effectuate the optimality principle also grows linearly with the processing times and grows exponentially with the number of machines. These time and space requirements seriously limit the applicability of the dynamic programming algorithm. In that sense, the parallel-machine problem seems to be much more difficult to solve than the first two problems we considered.

1.1.4. Problem classification

Because of the huge variety of machine scheduling problems, we need a classification scheme to make them rapidly accessible and easy to refer to. We adopt the notation and terminology of the classification scheme for deterministic machine scheduling problems as proposed by Graham, Lawler, Lenstra, and Rinnooy Kan [1979]. Classification takes place by use of a three-field notation $\alpha\,|\,\beta\,|\,\gamma$.

The first field α specifies the machine environment. For instance, 1 refers to the special case of a single machine and F denotes the flow-shop situation. In case of parallel machines, we have $\alpha\in\{P, Q, R\}$, since three cases can be distinguished. We have $\alpha=P$ if $p_{ij}=p_j$ for each J_j and M_i; in this case, the machines are said to be *identical*. If $p_{ij}=p_j/s_i$, where s_i denotes the *speed* of machine M_i, then the machines are *uniform*, which is denoted by Q. In the general case, the machines are *unrelated*, which is specified by R. In general, the number of machines is specified as part of the problem instance. However, if the symbols F, P, Q, or R, are immediately followed by an integer, then the number of machines is specified as part of the problem type and is equal to this integer. For example, $F2$ refers to the 2-machine flow shop.

The second field contains the job characteristics. If it is empty, then the default assumptions apply. This means that preemption of jobs is not allowed, that no precedence relations are specified, that $r_j=0$ and $d_j=\infty$ for all jobs, and that the processing times are arbitrary non-negative integers. The most common acronyms that occur in this field are *pmtn*, *prec*, r_j, and d_j, indicating that preemption

is allowed, that there are precedence relations between the jobs, that the jobs have release times, and that the jobs have deadlines, respectively.

The third field specifies the objective function. For instance, $\gamma = C_{\max}$ denotes the makespan criterion, and $\gamma = \Sigma_{j=1}^{n} w_j C_j$ means that the objective is to minimize the sum of the weighted job completion times. Other important objective functions are the sum of weighted tardiness $\Sigma_{j=1}^{n} w_j T_j$, where T_j denotes the tardiness of J_j, defined as $T_j = \max\{C_j - d_j, 0\}$, and the maximum lateness $L_{\max}$, defined as $L_{\max} = \max_{1 \leq j \leq n}(C_j - d_j)$. $L_j = C_j - d_j$ is called the lateness of J_j.

The three scheduling problems introduced earlier are denoted by $1 \| \Sigma w_j C_j$, $F2 \| C_{\max}$, and $R \| C_{\max}$, respectively.

1.2. Combinatorial optimization

Machine scheduling problems belong to the area of *combinatorial optimization*. Combinatorial optimization involves problems in which we have to choose the best from a finite number of relevant solutions. For the first two scheduling problems, for instance, we can restrict ourselves to the $n!$ permutations of the n jobs: for each permutation (or sequence) there is only one relevant schedule, since the schedules with avoidable machine idle time before or between the execution of jobs are dominated by the schedules without avoidable machine idle time. For the third problem, there are at most m^n assignments. Given an assignment of jobs to machines, we can easily compute the associated minimum makespan.

The finiteness of the solution set suggests the brute-force approach of *exhaustive* or *explicit enumeration* to be effective: simply generate all feasible solutions, examine their costs, and select the best one. Such an approach can be very time-consuming, since the required effort to examine all schedules grows exponentially with the number of jobs. It is to be expected that the number of basic arithmetic operations (additions, subtractions, and multiplications) to be performed will at least be of the order $n!$, $n!$, and m^n, respectively. As there is an upper bound on the number of operations that a computer can perform per period of time, problems of only very limited size are effectively solvable by explicit enumeration.

We have therefore good reasons to search for faster algorithms. Such algorithms are apparently available for the problems $1 \| \Sigma w_j C_j$ and $F2 \| C_{\max}$: both are solvable by simple scheduling rules that ask us to arrange the jobs in a certain order. The $R \| C_{\max}$ problem seems to be much harder to solve. Although under certain circumstances the dynamic programming algorithm may be preferred to complete enumeration, the effort to solve the problem compares very poorly with the effort needed for the first two problems. The fundamental question is whether there exists a simple algorithm for $R \| C_{\max}$ or not. If not, then this problem may considered to be 'hard' in comparison with the 'easy' problems $1 \| \Sigma w_j C_j$ and $F2 \| C_{\max}$.

The distinction between easy and hard problems apparently involves the effort required to solve them to optimality. Since the effort grows with the size of the problem instance, it makes sense to express the effort as some function of this size. The *size of an instance* is defined as the number of symbols required to

represent it. The size is *encoding*-dependent: it makes a difference which representation system we employ. Integers may be represented by an arithmetic system to some fixed base $B \geqslant 2$, in which case $\lceil \log_B n \rceil$ symbols are required to represent an integer n. If $B = 2$, then we have a *binary* encoding. If $B = 10$, then we have a *decimal* encoding. Another system is a *unary* encoding. Under a unary encoding, integers are represented by a series of 1's, the length of which is equal to the value of the integer: to represent an integer n, we need n symbols. We will see that the difference between a unary encoding and an arithmetic encoding to some fixed base is relevant for number problems. A problem is called a *number* problem if no polynomial function p exists such that for any instance I the largest integer occurring in I is bounded from above by the value that p assumes for the size of I.

For the representation of integers by arithmetic systems to some fixed base it does not really matter what base is used. Since B is fixed and $\log_B n = \log n / \log B$, the number of symbols required to represent an integer n grows as a logarithmic function of n for any $B \geqslant 2$. In the remainder, we only consider a binary encoding. The results, however, apply also to arithmetic systems to other fixed base $B > 2$.

The *running time* of an algorithm for a given problem is measured by an upper bound on the number of elementary steps that the algorithm performs on any valid input, expressed as a function of the size of the input. If the size of the instance is measured by n, then the running time of an algorithm is expressed as $O(f(n))$ if there are constants c and n_0 such that the number of steps for any problem instance with $n \geqslant n_0$ is bounded from above by $cf(n)$. A similar definition can be given for the *space requirement* of an algorithm. These measures express the *rate of growth* [Papadimitriou and Steiglitz, 1982] of the complexity of the algorithm.

If we reconsider $1 | | \Sigma w_j C_j$, then the size of a problem instance is the number of symbols required to represent p_j and w_j for $j = 1, \ldots, n$. This size is $O(\Sigma_{j=1}^{n} (\log p_j + \log w_j))$ under a binary encoding. However, we will assume throughout that each basic arithmetic operation requires constant time. We assert that complete enumeration can then be implemented to run in $O(n!)$ time.

DEFINITION 1.1. A problem is *solvable in polynomial time* with respect to a certain encoding if there exists an algorithm for it whose running time is bounded from above by a function that is a polynomial in the length of that encoding.

Hence, if the length of the encoding is measured by n, then an algorithm is a polynomial-time algorithm if its running time is $O(n^k)$, for some fixed k. From now on, a problem is said to be easy if it is solvable in polynomial time. The problems $1 | | \Sigma w_j C_j$ and $F2 | | C_{\max}$ are easy. Both Smith's and Johnson's rule require that the jobs be arranged in a certain order. Since sorting a list of n elements takes $O(n \log n)$ time (see e.g. Aho, Hopcroft, and Ullman [1982]), both problems are solved in $O(n \log n)$ time. Note that the space required is also polynomially bounded. In general, the space requirement is polynomially bounded if the running time is.

The dynamic programming algorithm for the $R \mid \mid C_{max}$ problem requires $O(nmC^m)$ time and $O(nC^m)$ space, where C is an upper bound on the optimal makespan. If the length of the encoding is measured in the obvious terms of the number of jobs n, the number of machines m, and $\log(\Sigma_{i=1}^m \Sigma_{j=1}^n p_{ij})$, then this is clearly not a polynomial procedure: the number of machines appears in the exponent and C cannot be bounded in the above terms.

1.2.1. Computational complexity: the classes $\mathcal{P}$ and $\mathcal{NP}$

There is an elegant theory of computational complexity that classifies problems as hard or easy. It carries strong evidence that it is very unlikely that there exist polynomial-time algorithms for problems that are classified as hard. For an introduction to this theory, we refer to the seminal works by Cook [1971] and Karp [1972], and to the textbook by Garey and Johnson [1979]. In this section, we confine ourselves to an informal description and a review of the basic concepts.

The theory involves decision problems rather than optimization problems. We define a *decision* problem to be a question to which the answer is either 'yes' or 'no'. Any optimization problem can be viewed as a finite series of decision problems. Suppose the objective is to minimize some function $f(x)$ over x; the associated decision problems are then of the type: is $f(x) \leqslant k$?, where k is repeatedly adjusted by binary search over an appropriate interval for k. If a particular decision problem is solvable in polynomial time, then the related optimization problem is solvable in polynomial time if its optimal solution value is an integer whose logarithm is polynomially bounded in the size of the input.

DEFINITION 1.2. The class $\mathcal{P}$ contains all the decision problems that are solvable in polynomial time.

The decision variant of the $1 \mid \mid \Sigma w_j C_j$ problem is as follows: given an integer k, is there a schedule for which $\Sigma_{j=1}^n w_j C_j \leqslant k$? If the answer is 'yes', then we can verify the correctness of the answer in polynomial time when provided with a schedule. After all, the schedule specifies the job completion times, which in turn serve as input for some algorithm that checks whether the schedule is feasible and whether $\Sigma_{j=1}^n w_j C_j \leqslant k$. Under a binary encoding, both the input and the running time of the algorithm can be polynomially bounded. The job completion times are then a *concise certificate* for a polynomial-time *certificate-checking algorithm*. The decision problems for $F \mid \mid C_{max}$ and $R \mid \mid C_{max}$ are of the same type: given an integer k, is there a schedule with $C_{max} \leqslant k$? It is easy to verify that concise certificates and polynomial certificate-checking algorithms exist for these problems.

At this point, we are ready to introduce the class $\mathcal{NP}$.

DEFINITION 1.3. The class $\mathcal{NP}$ comprises all decision problems for which concise certificates and polynomial-time certificate-checking algorithms exist.

Note that we have that $\mathcal{P} \subseteq \mathcal{NP}$. The crucial question is whether $\mathcal{P} = \mathcal{NP}$, that is,

are the problems in $\mathcal{NP}$ solvable in polynomial time? It is widely conjectured that this is not the case, i.e., that $\mathcal{P} \neq \mathcal{NP}$, since $\mathcal{NP}$ contains so many difficult problems, including the decision versions of the traveling salesman problem and integer programming, for which polynomial-time algorithms have not been found, in spite of the many man years devoted to these problems. The notion of polynomial reducibility is very important in complexity theory; we give the definition as formulated by Lawler, Lenstra, Rinnooy Kan, and Shmoys [1989].

DEFINITION 1.4. A problem A is *polynomially reducible* to problem B if and only if there exists a polynomial-time computable function τ that transforms inputs for A into inputs for B such that x is a 'yes' input for A if and only if $\tau(x)$ is a 'yes' input for B.

The notion of reducibility in polynomial time is transitive: if a problem A reduces polynomially to problem B, and if problem B reduces polynomially to problem C, then problem A reduces polynomially to problem C.

DEFINITION 1.5. A problem is said to be $\mathcal{NP}$-complete if it is a member of the class $\mathcal{NP}$, and if every problem in $\mathcal{NP}$ is polynomially reducible to it.

The notion of $\mathcal{NP}$-completeness interconnects the hardest problems in $\mathcal{NP}$ in the sense that if there would be a polynomial-time algorithm for one particular $\mathcal{NP}$-complete problem, then it could be transformed into polynomial-time algorithms for all other $\mathcal{NP}$-complete problems. In order to prove that a particular problem is $\mathcal{NP}$-complete, we must show that it is a member of $\mathcal{NP}$ and that all other problems in $\mathcal{NP}$ are polynomially reducible to it. Cook [1971] proves this for the so-called *satisfiability problem*. Since the notion of reducibility is transitive, it suffices for any other alleged $\mathcal{NP}$-complete problem to show that it is in $\mathcal{NP}$ and that some *known* $\mathcal{NP}$-complete problem polynomially reduces to it. The above definitions and concepts imply that it is very unlikely that there exist polynomial-time algorithms for problems that are $\mathcal{NP}$-complete.

Garey and Johnson [1979] present an extensive list of problems that have been shown to be $\mathcal{NP}$-complete, including the PARTITION problem.

PARTITION
Given a multiset $\mathcal{Q} = \{a_1, \ldots, a_t\}$ of t integers, does $\mathcal{Q}$ include a subset $\mathcal{Q}_1$ such that

$$\sum_{a_j \in \mathcal{Q}_1} a_j = \sum_{j=1}^{t} a_j / 2?$$

We now prove what we alluded to before: the decision problem of the $R \mid \mid C_{\max}$ problem is $\mathcal{NP}$-complete; hence, it is very unlikely that there exists a polynomial-time algorithm for the optimization problem. We give a polynomial reduction from PARTITION.

THEOREM 1.3. *The decision variant of the $R \mid \mid C_{\max}$ problem is $\mathcal{NP}$-complete, even in the case of two identical machines.*

PROOF. It is obvious that the problem is a member of the class $\mathcal{NP}$. We prove that PARTITION is polynomially reducible to our decision problem. For any instance of PARTITION, construct the following instance of the scheduling problem:

$$\begin{aligned} m &= 2, \\ n &= t, \\ p_{ij} &= a_j, \quad \text{for } i = 1, \ldots, m,\ j = 1, \ldots, n, \\ k &= \sum_{j=1}^{t} a_j / 2. \end{aligned}$$

It is easy to verify that the question whether there exists a schedule with $C_{\max} \leqslant k$ has an affirmative answer if and only the instance of the PARTITION problem is a 'yes' instance. Clearly, the reduction is polynomial. □

Optimization problems are not in $\mathcal{NP}$, but they are said to be *$\mathcal{NP}$-hard* if their decision variants are $\mathcal{NP}$-complete: apparently, such problems are at least as hard as the problems in $\mathcal{NP}$.

We come back to the difference between a binary and a unary encoding of the input. The distinction is relevant for those number problems that are $\mathcal{NP}$-complete under a binary encoding, but solvable in polynomial time under a unary encoding. For PARTITION, for instance, there is a dynamic programming algorithm that runs in $O(t\Sigma_{j=1}^{t} a_j)$ time and space, which is polynomially bounded under a unary encoding. Such an algorithm is not polynomial under a binary encoding, and is therefore called a *pseudo-polynomial-time* algorithm. Problems that are $\mathcal{NP}$-complete under both encodings are called *strongly* $\mathcal{NP}$-complete. Problems are said to be *ordinarily* $\mathcal{NP}$-complete if they are $\mathcal{NP}$-complete under a binary encoding.

Note that the $Rm \mid \mid C_{\max}$ problem is solvable in polynomial time under a unary encoding. The number of machines is here specified as part of the problem type and not of the problem instance. In other words, the number of machines is *fixed.* The dynamic programming algorithm runs in $O(nmC^m)$ time and $O(nC^m)$ space, with C some upper bound on the makespan. Since we have that $C \leqslant \min_{1 \leqslant i \leqslant m} \Sigma_{j=1}^{n} p_{ij}$, the running time can alternatively be written as $O(nm(\min_{1 \leqslant i \leqslant m} \Sigma_{j=1}^{n} p_{ij})^m)$ time. For fixed m, this time requirement is polynomial under a unary encoding.

Although we have shown that the decision variant of $R \mid \mid C_{\max}$ is $\mathcal{NP}$-complete in the ordinary sense for a fixed number of machines by a reduction from PARTITION, it is possible to prove that the problem is $\mathcal{NP}$-complete in the strong sense in case of an arbitrary number of machines. The reduction is from 3-PARTITION, one of the basic strongly $\mathcal{NP}$-complete problems.

3-PARTITION

Given an integer b and a multiset $\mathcal{A} = \{a_1, \ldots, a_{3t}\}$ of $3t$ positive integers with $\frac{1}{4}b < a_j < \frac{1}{2}b$ for each j, $j = 1, \ldots, 3t$, and with $\Sigma_{j=1}^{3t} a_j = tb$, is there a partition of $\mathcal{A}$ into t mutually disjoint subsets $\mathcal{A}_1, \ldots, \mathcal{A}_t$ such that

$$\mathcal{A} = \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_t$$

and

$$\sum_{a_j \in \mathcal{A}_1} a_j = \cdots = \sum_{a_j \in \mathcal{A}_t} a_j = b?$$

The reduction is omitted, but proceeds in the same spirit as the former.

1.2.2. Optimization

Although optimization algorithms for hard combinatorial optimization algorithms are unavoidably enumerative in nature, the aim is still to develop algorithms that perform satisfactorily well on the average for instances of reasonable size. The main concern is to avoid *exhaustive enumeration* of the solution space, since this would imply computational suicide. There are a number of generic optimization methods: dynamic programming, branch-and-bound, and cutting plane methods.

Both *dynamic programming* and *branch-and-bound* aim at *implicit enumeration* of the solution space. For the application of dynamic programming, we need to identify some underlying principle of optimality. We have seen an example of dynamic programming for the $R \,|\,|\, C_{\max}$ problem. Application of the optimality principle may require both time and space that is not bounded by a polynomial in the length of the input. Nonetheless, dynamic-programming based pseudo-polynomial algorithms may be very efficient.

Branch-and-bound solves a combinatorial optimization problem 'by breaking up the feasible set of solutions into successively smaller subsets, calculating bounds on the objective function value over each subset, and using them to discard certain subsets from further consideration' [Balas and Toth, 1985].

For the $R \,|\,|\, C_{\max}$ problem, for instance, a simple lower bound on the objective function over the entire set of solutions is given by $\lceil \Sigma_{j=1}^{n} \min_{1 \leqslant i \leqslant m} p_{ij} / m \rceil$, where $\lceil x \rceil$ denotes the smallest integer not smaller than x. After all, the n jobs combined require at least $\Sigma_{j=1}^{n} \min_{1 \leqslant i \leqslant m} p_{ij}$ time, and it is ideal to split this requirement equally over the m machines. This bound may be rounded up to the nearest integer, since the optimal makespan will be integral due to integrality of the processing times. Partitioning of the set of all feasible schedules into m subsets could proceed according to the assignment of some job. For the instance given in Table 1.3, we partition the feasible set into three subsets according to whether we assign J_1 to M_1, M_2, or M_3. Bearing in mind that 24 is an upper bound on the optimal makespan, we discard the assignment of J_1 to M_1, since this requires 30 units of time. Hence, we ignore all feasible solutions in which J_1 is scheduled on M_1. Now consider the scheduling of J_1 on M_2. A lower bound is then given by $(p_{12} + \Sigma_{j=2}^{8} \min_{1 \leqslant i \leqslant m} p_{ij}) / m = 21\frac{2}{3}$. As we cannot discard the assignment of J_1 to M_2 at this stage, we may further

partition this set of feasible solutions with J_1 scheduled on M_2 according to whether we assign J_2 to M_1, M_2, or M_3. We proceed in this way until we have discarded all possible subsets.

The feasible set of sequencing problems, including single-machine and flow-shop problems, is usually partitioned according to the job that is sequenced in the next available position, starting either at the beginning or at the end of the sequence.

It is helpful to visualize the partitioning of the feasible set as a *tree* turned upside down, having *nodes* and *branches*. The nodes represent the subsets; the branches indicate a further partitioning of these subsets.

Several factors affect the performance of a branch-and-bound algorithm. The growth of the search tree depends largely on the strength of the lower bound. The stronger the lower bound the fewer nodes have to be examined, but strong lower bounds usually ask more computing time than weaker bounds. The size of the tree is also affected by the quality of the upper bound, the use of other elimination criteria, and the partitioning (or *branching*) strategy. Furthermore, the strategy that dictates the order in which the nodes are examined is also very important.

Lagrangian relaxation (Section 1.3) is a generic technique to compute strong lower bounds; it may also be helpful for the development of upper bounds, elimination criteria, and partitioning strategies.

The aim of *cutting plane* methods, finally, is to solve integer linear programming problems by linear programming. This is possible if we have a complete description of the convex hull of the solution space by means of linear inequalities. In general, it is difficult to find such a description. A partial description, however, is also useful, since the solution of the linear program may then induce a strong lower bound, which in turn can be used in a branch-and-bound or branch-and-cut algorithm. The latter has been shown to be successful for a number of combinatorial optimization problems including the traveling salesman problem (see e.g. Grötschel and Padberg [1985] and Padberg and Rinaldi [1987]), but its success for machine scheduling problems is modest up to now. Cutting planes will not be considered here; we refer to Schrijver [1987] for an elaborate treatment of this subject.

1.2.3. Approximation

If a combinatorial optimization problem is $\mathcal{NP}$-hard, then we know that an optimization algorithm is likely to take a superpolynomial amount of time in the worst case. This makes optimization algorithms unreliable, since it is usually impossible to gauge beforehand how much time will be needed. Instead of pursuing an optimal solution, we could settle for a good *approximate* solution that hopefully can be obtained at the expense of considerably less effort. The dilemma is to spend an uncontrollable and possibly exponential amount of time to find an optimal solution, or to spend an acceptable amount of time to find a near-optimal solution. For the second option, we also have to deal with the trade-off between the quality of the approximate solution and the time invested to find it. How can we measure the quality of an approximate solution, or at a

higher level, how can we evaluate the performance of an approximation algorithm? There are a few methods for this, including *empirical* analysis, *probabilistic* analysis, and *worst-case* analysis.

The empirical analysis of an approximation algorithm can be conducted in various ways. Usually, it comes down to running a considerable number of tests on different instances of the problem, and comparing the approximate solution value with the solutions generated by other approximation algorithms, with the optimal solution value (if this can be found), or with a lower bound on the optimal value. By performing empirical analysis, the aim is to get some feeling of the average performance of the approximation algorithm.

For the $R \mid \mid C_{max}$ instance, we applied a simple *dispatching rule*: while running through the list of jobs, we assigned each job to the machine that has the smallest processing time for it, thereby obtaining a schedule with $C_{max} = 33$. Recall that a lower bound is given by $\lceil \Sigma_{j=1}^{n} \min_{1 \leqslant i \leqslant m} p_{ij} / m \rceil$, which gives 19 for the instance of Table 1.3. This renders little information about this instance: the approximate solution in Figure 1.3 is poor, or the lower bound is.

The probabilistic performance analysis of an approximation algorithm takes place subject to a chosen probability distribution of the instances, and all statements are made subject to this distribution. The goal is usually to establish that the algorithm is asymptotically optimal under a certain probability distribution of the instances. Another, equally important, aim is to get a probabilistic characterization of the optimal solution value. This, in itself, is often the basis for a probabilistic performance analysis of the algorithm.

A worst-case analysis measures the behavior of the algorithm in the worst case rather than in the average case. Such an analysis conveys a pessimistic view, since the worst case may not be representative. However, a worst-case analysis proceeds independently of the distribution of problem instances, and gives therefore a performance guarantee for all instances. An approximation algorithm that asymptotically never delivers a solution value of more than ρ times the optimal solution value is called a ρ-approximation algorithm. We refer to ρ as the *worst-case* ratio.

The dispatching rule for the $R \mid \mid C_{max}$ problem has worst-case ratio m. After all, we have that the resulting makespan is no more than $\Sigma_{j=1}^{n} \min_{1 \leqslant i \leqslant m} p_{ij}$, and a lower bound is given by $\Sigma_{j=1}^{n} \min_{1 \leqslant i \leqslant m} p_{ij} / m$. To show that this ratio is tight, consider the following instance:

$$
\begin{aligned}
n &= km, \\
p_{1j} &= 1, \qquad \text{for } j = 1, \ldots, n, \\
p_{ij} &= 1 + \epsilon, \quad \text{for } i = 2, \ldots, m, \; j = 1, \ldots, n,
\end{aligned}
$$

where k is some given constant, and $0 < \epsilon < 1/k$. The approximation algorithm assigns all jobs to M_1, thereby producing a schedule with makespan n. In any optimal schedule, however, exactly k jobs are assigned to each machine; this gives the makespan $k(1+\epsilon)$. It is easy to see that the ratio $\rho = n/(k + k\epsilon) \to m$ if $\epsilon \to 0$.

A number of popular techniques are applicable to the design of approximation

algorithms for machine scheduling problems. We will only give a sample of these techniques, not a complete classification. We have already seen a member of a simple but widely applicable class of approximation algorithms: the class of *dispatching rules*. These rules make use of priority functions that associate an urgency measure with each job, according to which it is assigned or sequenced. Such priority functions are based upon intuitively reasonable but often only locally valid arguments that estimate the urgencies of the jobs. Not surprisingly, they may produce myopic and erroneous schedules. Dispatching rules are nonetheless widely applied in complex and large job-shop systems. On the one hand, they are easy to develop and to implement no matter the problem setting; on the other hand, their robustness tends to grow with the size of the problem.

The second class contains the approximation algorithms based upon *dynamic programming and rounding*. This technique is only applicable to a problem if there is a pseudo-polynomial algorithm available for its solution. The central idea is not to consider the entire state space, but only a specific part of it. In fact, this part is chosen such that the dynamic programming, running in pseudo-polynomial time on the entire state space, runs in polynomial time on the reduced state space. Such algorithms usually have a performance guarantee. This is achieved by rounding down all the data of the problem to a multiple of the desired accuracy $\rho > 1$, by which only a limited number of distinct data remain, and by running the dynamic programming-based procedure. For the $R \mid \mid C_{max}$ problem, for instance, Horowitz and Sahni [1976] apply this principle to develop a ρ-approximation algorithm, running in $O(nm(nm/(\rho-1))^{m-1})$ time and space.

Third, there are the so-called *truncated* branch-and-bound algorithms. The idea here is to discard some of the feasible subsets, even if there is no valid reason to do so. The decision what subsets to discard can be made arbitrarily; for instance, termination may occur upon reaching a prespecified number of nodes. It makes more sense, however, to develop a more involved strategy for this. The following strategy gives rise to an approximation algorithm with a performance guarantee. Instead of discarding a node if the associated lower bound LB is no less than the upper bound UB, we now discard a node if $\rho LB \geqslant UB$ for some predetermined $\rho > 1$. This is a ρ-approximation algorithm, but, in general, it is still impossible to bound its running time by a polynomial in the size of the input.

Local-search algorithms, finally, are two-phase approximation algorithms. In the first phase, a schedule is generated that serves as input for the second phase; in the second phase, the schedule is adjusted somewhat in order to improve on its value. For the second phase, the usual procedure is to define for a given schedule σ a *neighborhood* N_σ as the set of schedules that can be obtained from σ by carrying out a prespecified type of changing operations. For parallel-machine scheduling problems, for instance, we could define the neighborhood of the schedule σ as the set of schedules that are obtained either by reassigning one job, or by swapping two jobs that are scheduled on different machines in σ. For sequencing problems, like single-machine and flow-shop problems, it is customary to define the neighborhood of a schedule as those schedules that are obtained either by

repositioning one job in the sequence, or by interchanging two jobs. The procedure proceeds by searching the neighborhood N_σ for a schedule with smaller objective value, which will be our new approximate solution. This process is repeated and terminates when no further improvement can be found. Evidently, procedures based upon this concept provide locally optimal solutions; in general, they cannot be guaranteed to find globally optimal solutions.

Although the procedure above is typical of a plain local-search algorithm, the main danger is to get trapped in a relatively poor local optimum. We may circumvent this pitfall by using multiple schedules as starting points, which may lead us to multiple locally optimal solutions. Hopefully, one of them is a good approximate solution.

More sophisticated techniques to avoid early entrapment have been developed, among which *simulated annealing* and *tabu search* take prominent places. Simulated annealing (see e.g. Van Laarhoven and Aarts [1987]) leaves the possibility open to travel from one local optimum to another. This is achieved by accepting deteriorations of the objective value with a probability that is a decreasing function of running time. Tabu search [Glover, 1989; De Werra and Hertz, 1989] is much similar to simulated annealing, but provides a deterministic mechanism to accept deteriorations.

Globally, we can say that local-search algorithms are easy to develop and to implement, and are known to produce excellent results.

1.3. Lagrangian relaxation and duality

In Section 1.2, we mentioned the need for strong lower bounds if we want to apply branch-and-bound. Mathematical formulations often provide the insight to derive good lower bounds. A common strategy is to relax some of the constraints such that the resulting problem is easier to solve and provides a lower bound to the original problem. The simplest way is the *linear programming relaxation*: formulate the problem as an integer linear programming problem, drop the integrality constraints on the variables, and then solve the linear programming relaxation. Other relaxation methods, such as *Lagrangian relaxation* and *surrogate relaxation*, are more intricate to perform, but give lower bounds that are theoretically at least as good. In this thesis, we consider Lagrangian relaxation and Lagrangian duality, and their opportunities for the development of optimization and approximation algorithms for machine scheduling problems. Although it is common practice to speak about the Lagrangian relaxation of a combinatorial optimization problem, it is more correct to refer to the Lagrangian relaxation of a particular formulation of the problem. An optimization problem often allows different formulations, and one formulation may offer completely different opportunities than others.

Lagrangian relaxation is already a conventional technique, dating back to the work by Held and Karp [1970, 1971] on the traveling salesman problem. Since then, it has shown its merits for a gamut of hard combinatorial optimization problems, running from the traveling salesman problem, particular plant location and machine scheduling problems, to general integer linear programming problems. Excellent introductions to Lagrangian relaxation are given by

Geoffrion [1974A], Shapiro [1979] and Fisher [1981]; an overview of its applications is given by Fisher [1981, 1985].

Traditionally, though, the emphasis has been on problems that are formulated in terms of integer (usually 0-1) linear programming problems. In this section, we give a broader introduction to the basic concepts of Lagrangian relaxation than found in the literature referred to earlier, since we do not confine ourselves to problems that are formulated as 0-1 linear programs. It will be oriented towards machine scheduling, and will place earlier applications of Lagrangian relaxation to machine scheduling problems in a specific context. It is unavoidable, however, that our introduction partly parallels other expositions. The undertone of this thesis is that Lagrangian relaxation has a wider range of applications than found in and suggested by the literature. We do consider other formulations, and, in the subsequent chapters, we show that they facilitate the application of Lagrangian relaxation to machine scheduling problems.

To show that combinatorial optimization problems allow different formulations, and to demonstrate what we mean by formulations other than integer linear programming formulations, we give three different formulations for the $1 | | \Sigma w_j C_j$ problem.

The first one is an integer linear programming formulation, an extension of which is employed by Potts [1985A] and Dyer and Wolsey [1990] to obtain lower bounds for the problems $1 | prec | \Sigma w_j C_j$ and $1 | r_j | \Sigma w_j C_j$, respectively. We introduce 0-1 decision variables x_{jk} $(j, k = 1, \ldots, n)$ that assume the value 1 if job J_j is sequenced before J_k, and the value 0 otherwise. Naturally, we must have $x_{jj} = 0$, for $j = 1, \ldots, n$. The completion time of J_j is then given by $\Sigma_{k=1}^{n} p_k x_{kj} + p_j$. The problem is to minimize

$$\sum_{j=1}^{n} \sum_{k=1}^{n} w_j p_k x_{kj} + \sum_{j=1}^{n} w_j p_j$$

subject to

$$x_{kj} + x_{jk} = 1, \qquad \text{for } j, k = 1, \ldots, n, j \neq k, \tag{1.3}$$

$$x_{kj} + x_{lk} + x_{jl} \geqslant 1, \qquad \text{for } j, k, l = 1, \ldots, n, j \neq k, j \neq l, k \neq l, \tag{1.4}$$

$$x_{jk} \in \{0, 1\}, \qquad \text{for } j, k = 1, \ldots, n, \tag{1.5}$$

$$x_{jj} = 0, \qquad \text{for } j = 1, \ldots, n. \tag{1.6}$$

The constraints (1.3) ensure that J_j is sequenced either before J_k or after J_k. The conditions (1.4) are transitivity constraints that disallow cycles: if J_j is sequenced before J_k and J_k before J_l, then we cannot have J_j sequenced after J_l.

The second formulation is a generic one; extensions are found for multiple-machine problems (see e.g. Thompson and Zawack [1985/6]). We define $T = \Sigma_{j=1}^{n} p_j$. We now introduce integer variables x_{jt} that take the value 1 if J_j starts at time t, and the value 0 otherwise. The $1 | | \Sigma w_j C_j$ problem can then be formulated as to minimize

$$\sum_{j=1}^{n} \sum_{t=0}^{T-1} w_j (t + p_j) x_{jt}$$

subject to

$$\sum_{t=0}^{T-1} x_{jt} = 1, \qquad \text{for } j = 1, \ldots, n, \tag{1.7}$$

$$\sum_{j=1}^{n} \sum_{s=\max\{t-p_j,0\}}^{t-1} x_{js} = 1, \quad \text{for } t = 0, \ldots, T-1, \tag{1.8}$$

$$x_{jt} \in \{0, 1\}, \qquad \text{for } j = 1, \ldots, n, \ t = 0, \ldots, T-1. \tag{1.9}$$

The conditions (1.7) ensure that each job is started exactly once in the interval $[0, T-1]$, the conditions (1.8) reflect that the machine can handle only one job at any point in time between 0 and T, and the conditions (1.9) are the integrality constraints on the variables. The major handicap of this formulation is the number of variables: we have nT of them. This formulation requires a pseudo-polynomial number of variables and constraints.

We now give a formulation that is not an integer linear programming formulation. Let C_j be the completion time of J_j, for $j = 1, \ldots, n$. The $1 \| \Sigma w_j C_j$ problem is then to determine job completion times $C_1, \ldots, C_n$ that minimize

$$\sum_{j=1}^{n} w_j C_j$$

subject to

$$C_k \geqslant C_j + p_k \ \text{ or } \ C_j \geqslant C_k + p_j, \qquad \text{for } j, k = 1, \ldots, n, j \neq k, \tag{1.10}$$

$$C_j - p_j \geqslant 0, \qquad \text{for } j = 1, \ldots, n. \tag{1.11}$$

The constraints (1.10) stipulate that the machine handles at most one job at a time: for every pair of jobs J_j and J_k, it must be that J_j precedes J_k or that J_k precedes J_j. The conditions (1.11) express the availability of the machine: no job can be started before time 0.

The tricky issue is actually the formulation of the capacity constraints of the machine. In the integer programming formulations, these capacity constraints are formulated by means of 0-1 variables, which accounts for the relatively large number of variables. In the third formulation, the capacity constraints are formulated by means of the 'logical' disjunctive constraints (1.10). It is important to realize that, in spite of the glossy formulations, each one still represents a polynomially solvable problem.

Such formulations provide the basis for formulations of hard problems. For instance, suppose we impose precedence constraints between the jobs. It is convenient to represent such constraints by an acyclic precedence graph G with vertex set $\{J_1, \ldots, J_n\}$ and arc set A, which equals its transitive reduction; i.e., no arc in A can be removed on the basis of transitivity. A path in G from J_j to J_k implies that J_j has to be sequenced before J_k. The $1|prec|\Sigma w_j C_j$ problem has been shown to be $\mathcal{NP}$-hard in the strong sense by Lawler [1978] and Lenstra and Rinnooy Kan [1978].

The precedence constraints require the addition of the following type of constraint in the first formulation,

$$x_{jk} = 1, \qquad \text{for each } (J_j, J_k) \in A,$$

of the following type in the second formulation,

$$\sum_{s=0}^{t} x_{js} - \sum_{s=0}^{t} x_{ks} \geqslant 0, \quad \text{for } t = 0, \ldots, T-1, \ \text{for each } (J_j, J_k) \in A,$$

and the following conditions in the third formulation,

$$C_k \geqslant C_j + p_k, \qquad \text{for each } (J_j, J_k) \in A.$$

Lower bounds for the $1|prec|\Sigma w_j C_j$ problem can be obtained by disregarding the precedence constraints. Considering the first two formulations, we can also obtain them by dropping the integrality constraints on the 0-1 variables, and by subsequently solving the resulting linear programming problems. For the second formulation, this approach is only feasible if T is not too large. At this point, it is not clear how the third formulation can be of use. This will become apparent in the next section, where we introduce the concept of Lagrangian relaxation.

1.3.1. Basic concepts of Lagrangian relaxation

The main idea behind Lagrangian relaxation is to see an $\mathcal{NP}$-hard combinatorial optimization problem as an 'easy-to-solve' problem complicated by a number of 'nasty' side constraints. These nasty side constraints are removed from the set of constraints, and put into the objective function, each weighted by a given Lagrangian multiplier. This is what we refer to as *dualizing* the nasty constraints. Dualizing the nasty constraints, we get an easy-to-solve problem, and its solution provides a lower bound on the optimal solution value of the original problem.

Examining the $1|prec|\Sigma w_j C_j$ problem, we may identify the precedence constraints as the nasty constraints. However, Potts [1985A], using the first formulation for the $1|prec|\Sigma w_j C_j$ problem, does not particularly focus on the precedence constraints when dualizing. For a specific formulation of the problem, there may be no unique set of nasty constraints. In addition, whether constraints are nasty or not can often only be decided for a specific formulation of the problem. This gives rise to competing formulations, and to competing relaxations of the same formulation. In Section 1.3.2, we present an example of the latter.

Consider the typical formulation for a combinatorial optimization problem, referred to as problem (P): minimize

$$cx \tag{P}$$

subject to

$$Ax \geqslant b,$$

$$x \in X,$$

where A is a given $m \times n$ matrix, b a given $m \times 1$ vector, and c is a given $1 \times n$

vector; x is an $n \times 1$ vector of decision variables. It is assumed that X is a non-empty closed set and that X has some computationally convenient structure not shared by the entire problem. We denote by $v(\mathrm{P})$ the optimal objective value of problem (P). If we now introduce a non-negative vector of Lagrangian multipliers $\lambda = (\lambda_1, \ldots, \lambda_m)$ to dualize the constraints $Ax \geqslant b$ with, we obtain the *Lagrangian problem* (L_λ). This problem is to find the value $L(\lambda)$, which for a given $\lambda \geqslant 0$ is the minimum of

$$(c - \lambda A)x + \lambda b \qquad (\mathrm{L}_\lambda)$$

subject to

$$x \in X.$$

Since we have assumed that X possesses some convenient structure, we may assume that the Lagrangian problem is easier to solve than the original problem.

THEOREM 1.4. *The value $L(\lambda)$ is a lower bound on $v(\mathrm{P})$ for any $\lambda \geqslant 0$.*

PROOF. Let x^* be an optimal solution to problem (P). Since $\lambda \geqslant 0$ and $Ax^* \geqslant b$, we have that

$$L(\lambda) \leqslant (c - \lambda A)x^* + \lambda b = cx^* + \lambda(b - Ax^*) \leqslant v(\mathrm{P}). \qquad \square$$

It is easy to verify that if the dualized constraints are equality constraints of the type $Ax = b$, then the associated vector of Lagrangian multipliers is unrestricted in sign. We assume throughout that the dualized constraints are inequalities. This assumption can be made without loss of generality, since any system of linear equations can be rewritten in terms of linear inequalities.

Since $L(\lambda)$ provides a lower bound on $v(\mathrm{P})$, we are interested in determining the Lagrangian multiplier λ^* that yields the best lower bound. This problem is called the *Lagrangian dual* problem, referred to as problem (D): maximize

$$L(\lambda) \qquad (\mathrm{D})$$

subject to

$$\lambda \geqslant 0.$$

In general, we cannot guarantee to have $v(\mathrm{P}) = L(\lambda^*)$; the difference $v(\mathrm{P}) - L(\lambda^*)$ is referred to as the *duality gap*. In contrast to problem (D), problem (P) is called the *primal* problem. If (P) is an integer linear programming problem, then $(\bar{\mathrm{P}})$ denotes its linear programming relaxation; $v(\bar{\mathrm{P}})$ denotes the optimal objective value for $(\bar{\mathrm{P}})$. As a matter of convenience, we use also an alternative notation for mathematical programming problems. For instance, we write problem (P) as $v(\mathrm{P}) = \min\{cx \mid Ax \leqslant b, x \in X\}$.

If $X = \{x \mid x \geqslant 0\}$, then problem (D) boils down to the dual problem in linear programming. Consider

$$L(\lambda^*) = \max\{\min\{(c - \lambda A)x + \lambda b \mid x \geqslant 0\} \mid \lambda \geqslant 0\}.$$

If the vector $(c - \lambda A)$ would be negative in one of its components, then the minimization over x would require that we put the corresponding component of x equal to $+\infty$, thereby making the lower bound worthless. We may confine ourselves, therefore, to those vectors λ for which $c - \lambda A \geqslant 0$. This gives

$$L(\lambda^*) = \max\{\lambda b \mid c - \lambda A \geqslant 0, \lambda \geqslant 0\},$$

which is clearly the dual of the linear program

$$\min\{cx \mid Ax \geqslant b, x \geqslant 0\}.$$

In the proof of the following theorem we make use of these arguments.

THEOREM 1.5. *If* (P) *is an integer linear programming problem, then* $L(\lambda^*) \geqslant v(\bar{\mathrm{P}})$.

PROOF. Without loss of generality, we may assume that $X = \{x \mid Dx \geqslant e, x \geqslant 0, x \text{ integral}\}$ with D a matrix and e a column vector, both of appropriate dimensions. We now have that

$$\begin{aligned} L(\lambda^*) &= \max\{\min\{(c - \lambda A)x + \lambda b \mid x \in X\} \mid \lambda \geqslant 0\} \\ &\geqslant \max\{\min\{(c - \lambda A)x + \lambda b \mid Dx \geqslant e, x \geqslant 0\} \mid \lambda \geqslant 0\} \\ &= \max\{\min\{(c - \lambda A - \mu D)x + \lambda b + \mu e \mid x \geqslant 0\} \mid \lambda \geqslant 0, \mu \geqslant 0\}\} \\ &= \max\{\lambda b + \mu e \mid c - \lambda A - \mu D \geqslant 0, \lambda \geqslant 0, \mu \geqslant 0\} \\ &= \min\{cx \mid Ax \geqslant b, Dx \geqslant e, x \geqslant 0\} \\ &= v(\bar{\mathrm{P}}). \end{aligned}$$

□

The proof of Theorem 1.5 indicates a sufficient condition for having $L(\lambda^*) = v(\bar{\mathrm{P}})$.

COROLLARY 1.1. *If* (P) *is an integer linear programming problem, and the problem* $\min\{cx \mid x \in X\}$ *can be solved as a linear programming problem for any given* $1 \times n$ *vector* c, *i.e., the integrality constraints on* x *are redundant, then we have that* $L(\lambda^*) = v(\bar{\mathrm{P}})$. □

Adopting the terminology of Geoffrion [1974B], we say that the Lagrangian problem possesses the *integrality property* if the integrality constraints on x are redundant.

1.3.2. An example: the generalized assignment problem

The *generalized assignment problem* is a useful problem for expository purposes because of its rich underlying structure. This accounts no doubt for the number of papers on the problem, in which various relaxations and variants of existing relaxations are proposed [Klastorin, 1979; Fisher, Jaikumar, and Van Wassenhove, 1986; Jörnsten and Näsberg, 1986; Sarin and Karwan, 1987; Karwan and Ram, 1987; Guignard and Rosenwein 1989, 1990; Barcia and

Jörnsten, 1990]. The generalized assignment problem has many interpretations, and therefore many potential applications. We interpret it as a machine scheduling problem closely related to the $R \mid \mid C_{\max}$ problem (see Section 1.1.3). Suppose we have a set of n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and a set of m unrelated parallel machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each of the jobs has to be scheduled on one of the machines. The processing of J_j on machine M_i requires an uninterrupted time p_{ij} for which a penalty $c_{ij} > 0$ is inflicted, for $i = 1, \ldots, m$, $j = 1, \ldots, n$. Each M_i $(i = 1, \ldots, m)$ can handle at most one job at a time, and is available for processing from time 0 up to time b_i: the total time required by the jobs assigned to it may not exceed b_i. The objective is to find a schedule of minimum total cost.

It is easy to verify that this problem is $\mathcal{NP}$-hard. The proof proceeds by a reduction from PARTITION, and is similar to the one for the $R \mid \mid C_{\max}$ problem. In the same spirit as for the $R \mid \mid C_{\max}$ problem, we can develop a dynamic programming algorithm for it that requires $O(nmb_{\max}^m)$ time and space, where $b_{\max} = \max_{1 \leqslant i \leqslant m} b_i$.

We formulate the problem as an integer program in the following way. We introduce variables x_{ij} that assume the value 1 if J_j is assigned to M_i, and the value 0 otherwise, for $i = 1, \ldots, m, j = 1, \ldots, n$. The problem is then to minimize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^{n} p_{ij} x_{ij} \leqslant b_i, \qquad \text{for } i = 1, \ldots, m, \tag{1.12}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad \text{for } j = 1, \ldots, n, \tag{1.13}$$

$$x_{ij} \in \{0, 1\}, \qquad \text{for } i = 1, \ldots, m, j = 1, \ldots, n. \tag{1.14}$$

The constraints (1.12) in this formulation make sure that the capacities of the machines are not exceeded, the constraints (1.13) enforce that each job is assigned, and the constraints (1.14) prohibit preemption. From a different angle, we can say that the constraints (1.12) are the knapsack constraints, the constraints (1.13) are the assignment constraints, and the constraints (1.14) are the integrality constraints.

First, we focus on the relaxation of the constraints (1.13). At first sight, it seems that the vector associated with them must be unsigned. However, since the equality constraints can without any consequence be replaced with the inequality constraints $\Sigma_{i=1}^{m} x_{ij} \geqslant 1$, we introduce a non-negative vector $\mu = (\mu_1, \ldots, \mu_n)$ of Lagrangian multipliers to dualize the constraints (1.13). We obtain the following Lagrangian problem: minimize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} - \mu_j) x_{ij} + \sum_{j=1}^{n} \mu_j$$

subject to (1.12) and (1.14), for a given $\mu \geqslant 0$.

This problem reduces to m knapsack problems, each of which can be solved in pseudo-polynomial time: solving the problem for machine M_i requires $O(nb_i)$ time. Hence, the Lagrangian is easy relative to the original problem.

Another option is to introduce a non-negative vector $\lambda = (\lambda_1, \ldots, \lambda_m)$ of Lagrangian multipliers and to dualize the constraints (1.12). In this fashion, we obtain another Lagrangian problem, which is to minimize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} + \lambda_i p_{ij}) x_{ij} - \sum_{i=1}^{m} \lambda_i b_i$$

subject to (1.13) and (1.14), for a given $\lambda \geqslant 0$.

This Lagrangian problem is solvable in $O(nm)$ time: assign each J_j to the machine that has minimum *dual cost* $c_{ij} + \lambda_i p_{ij}$ for it among the m machines. We note that this problem has the integrality property, implying that the best bound in this framework equals the optimal value of the linear programming relaxation. The latter is obtained by replacing the integrality constraints (1.14) by the conditions $x_{ij} \geqslant 0$, thereby allowing preemption.

1.3.3. Solving the Lagrangian dual problem

Since we have an interest in finding the best lower bound, we pursue the optimal Lagrangian multiplier λ^*. Two fundamental questions arise: (i) how can we find λ^*?, and (ii) how time-consuming is it to find λ^*? The second question reduces to the common trade-off between quality and speed. On the one hand, λ^* yields the best bound, but may be time-consuming to compute. On the other hand, it may be easier to compute an approximate value for λ^*, but this gives a lower bound of lesser quality.

An important observation is that the Lagrangian dual problem (D) is actually the problem of maximizing a linear function subject to a finite number of linear constraints. Since we have assumed problem (P) to be a combinatorial optimization problem, X contains only a finite number of relevant solutions. Hence, X can be represented as $X = \{x^{(1)}, \ldots, x^{(K)}\}$. Problem (D) is then equivalent to the following problem: maximize

$$z$$

subject to

$$z \leqslant (c - \lambda A)x^{(k)} + \lambda b, \quad \text{for } k = 1, \ldots, K,$$

$$\lambda \geqslant 0.$$

The reformulation makes it clear that the function $L : \lambda \rightarrow L(\lambda)$ is the lower envelope of a finite set of linear functions. We have depicted in Figure 1.4 the shape of L for $m = 1$ and $K = 5$.

The reformulation also indicates that (D) is solvable to optimality through techniques for linear programming problems with constraints given implicitly. In Section 1.3.3.1, we prove that, under certain conditions, problem (D) can be solved in polynomial time through Khachiyan's ellipsoid method [Khachiyan,

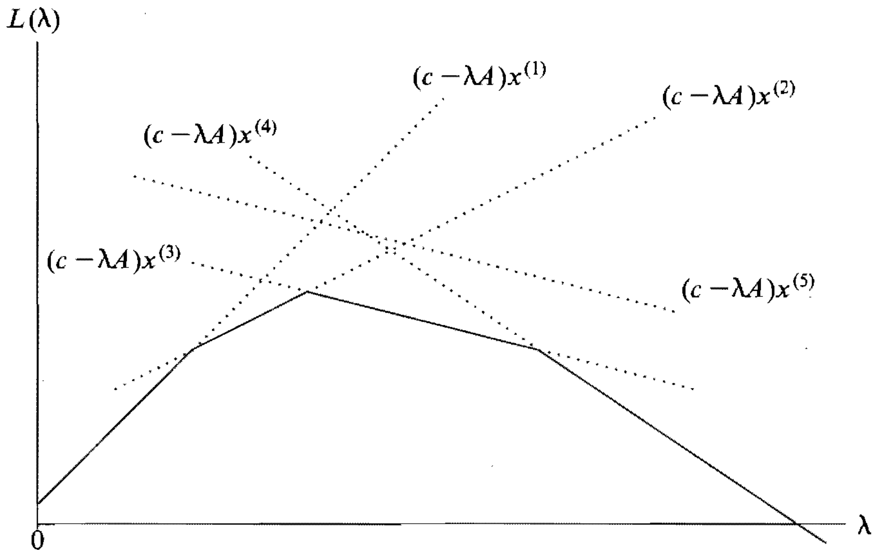


FIGURE 1.4. The form of $L(\lambda)$.

1979]. General techniques to solve problem (D) are the *subgradient method* and the *column generation* method. The former is easy to implement, and therefore frequently applied; it is described in Section 1.3.3.2. The latter is not discussed here, since it is rarely used; the method is difficult to implement and very slow in practice. For an exposition of these methods in a general context, we refer to the textbooks by Minoux [1986], Schrijver [1987], and Nemhauser and Wolsey [1988].

Any $\lambda \geqslant 0$ induces a lower bound on $v(\mathrm{P})$. Approximation algorithms for problem (D) are therefore usually local search methods. We discuss two types: *ascent direction* methods (Section 1.3.3.3) and *one-shot* methods (Section 1.3.3.4). In Section 1.3.3.5, we consider some miscellaneous approaches.

1.3.3.1. The ellipsoid method

The ellipsoid method is a polynomial algorithm for linear programming [Khachiyan, 1979]. We prove here that, under certain conditions, it can be applied to problem (D). The applicability of the ellipsoid method is only of theoretical interest; in practice, it is a very slow algorithm that is never used.

THEOREM 1.6. *Problem* (D) *is solvable in polynomial time if the problem* $\min\{\mu x \mid x \in X\}$ *is solvable in polynomial time for any* $1 \times n$ *vector* μ.

PROOF. Let $\mathcal{K} = \{(z,\lambda) \mid z \in \mathbb{R}, \lambda \in \mathbb{R}^m, z \leqslant (c-\lambda A)x^{(k)} + \lambda b$, for $k=1,\ldots,K$, $\lambda \geqslant 0\}$. To prove that the ellipsoid method is applicable to problem (D), it suffices to show that the following *separation problem for* $\mathcal{K}$ is solvable in polynomial time (see Grötschel, Lovász, and Schrijver [1981] and Padberg and Rao [1982]): given $(\bar{z},\bar{\lambda}) \in \mathbb{Q} \times \mathbb{Q}^m$, decide whether $(\bar{z},\bar{\lambda}) \in \mathcal{K}$; if not, give a *separating*

hyperplane, that is, an inequality $\lambda d^{\mathrm{T}} + \alpha z \leqslant \beta$ (d^{T} stands for the transpose of the row vector d), such that

$$(z,\lambda) \in \mathcal{K} \Rightarrow \lambda d^{\mathrm{T}} + \alpha z \leqslant \beta,$$

and

$$\bar{\lambda} d^{\mathrm{T}} + \alpha \bar{z} > \beta.$$

Observe now that for a given $(\bar{z},\bar{\lambda})$ we can determine the value $L(\bar{\lambda})$ and the corresponding solution vector $\bar{x}$ in polynomial time, since we have assumed that our subproblem is solvable in polynomial time. If $L(\bar{\lambda}) \geqslant \bar{z}$, then $(\bar{z},\bar{\lambda}) \in \mathcal{K}$; if $L(\bar{\lambda}) < \bar{z}$, then $(\bar{z},\bar{\lambda}) \notin \mathcal{K}$, and

$$\bar{z} > (c - \bar{\lambda} A)\bar{x} + \bar{\lambda} b$$

is a separating hyperplane. □

1.3.3.2. The subgradient method

We have seen that the function $L : \lambda \rightarrow L(\lambda)$ is the lower envelope of a finite set of linear functions; hence, L is continuous, piecewise linear, and concave in λ, but, unfortunately, not everywhere differentiable. The function L is not differentiable at any λ where (L_λ) has more than one optimal solution. We call such λ the *points of non-differentiability* of the function L; for $m = 1$, they are also called the *breakpoints* of L. However, the function L is everywhere *subdifferentiable*. A vector $\delta \in \mathbb{R}^m$ is called a *subgradient* of L at λ if it satisfies

$$L(\lambda + u) - L(\lambda) \leqslant u\delta, \quad \text{for each } u \in \mathbb{R}^m.$$

Hence, λ is optimal if and only if 0 is a subgradient of L at λ. In general, it is impossible to prove whether 0 is a subgradient, and we may only establish optimality if a sufficient condition for optimality (like the complementary slackness condition) is satisfied. The above suggests, nonetheless, that the solution for problem (D) can be approximated by an iterative procedure that generates a series of vectors of Lagrangian multipliers by moving a specified step size along a subgradient vector. If $\bar{x}$ is an optimal solution of problem ($\mathrm{L}_{\bar{\lambda}}$), then the vector $(b - A\bar{x})$ is such a subgradient at $\bar{\lambda}$. After all, for each $u \in \mathbb{R}^m$ we have that

$$\begin{aligned} L(\bar{\lambda} + u) - L(\bar{\lambda}) &\leqslant (c - (\bar{\lambda} + u)A)\bar{x} + (\bar{\lambda} + u)b - (c - \bar{\lambda})A\bar{x} + \bar{\lambda} b \\ &= u(b - A\bar{x}). \end{aligned}$$

This observation is the core of the subgradient method for solving problem (D). Let $\lambda^{(t)}$ denote the vector of Lagrangian multipliers after the tth iteration, and let $x^{(t)}$ be an optimal solution to problem ($\mathrm{L}_{\lambda^{(t)}}$). If $\lambda^{(0)}$ is the initial vector, then a series of vectors is generated by the following rule:

$$\lambda^{(t+1)} = \lambda^{(t)} + s^{(t)}(b - Ax^{(t)}),$$

where $s^{(t)} > 0$ is some scalar step size. The theoretical conditions for convergence of $\lambda^{(t)}$ to λ^* are that $s^{(t)} \rightarrow 0$ and that $\Sigma_{i=0}^{t} s^{(i)} \rightarrow \infty$ [Polyak, 1967]. These stringent conditions can of course not be observed in practice. However, for the

step size

$$s^{(t)} = \frac{\mu^{(t)}[z - L(\lambda)]}{\|Ax^{(t)} - b\|^2}$$

with $z \leq L(\lambda^*)$ and $0 < \mu^{(t)} \leqslant 2$ for each t, $\lambda^{(t)}$ converges either to λ^*, or to a vector $\bar{\lambda}$ such that $L(\bar{\lambda}) > z$ [Polyak, 1969; Held and Karp, 1971]. There are many rules to choose the sequence $\mu^{(t)}$ and the value z [Held, Wolfe, and Crowder, 1974; Camerini, Fratta, and Maffioli, 1975; Bazaraa and Goode, 1979; Karwan and Sarin, 1987]; usually, they are problem-specific and have an empirical justification.

In practice, the subgradient method is a non-polynomial approximation method. We are unable to establish whether it has converged to the optimal vector of Lagrangian multipliers. The main handicap of the subgradient method, however, is that it does not produce a series of monotonically increasing lower bounds. It is known for its zig-zagging in the beginning and slow convergence at the end (see, e.g., Held, Wolfe, and Crowder [1974]). The method is usually terminated when there is no significant increase of the objective value, or upon reaching a predetermined number of iterations. The subgradient method is undoubtedly the most frequently applied technique, since it is easy to implement. Furthermore, the ample computational experiments with the subgradient method reported in the literature give the impression that it produces invariably good approximate solutions.

1.3.3.3. Ascent direction methods

Considering the handicap of the subgradient method, it is natural to look for an iterative method that generates a series of monotonically increasing lower bounds. The key feature of such a method is the adjustment of only a limited number of multipliers per iteration, as opposed to the subgradient method where all multipliers are adjusted simultaneously. If the consequences of particular multiplier adjustments can be evaluated, then one guaranteeing an improved lower bound is chosen. Such methods are often referred to as *multiplier adjustment* methods, but we prefer to call them *ascent direction* methods.

An ascent direction method is problem-specific: it exploits the special structure of the problem and of the formulation. In general, an ascent direction method requires typically significantly less iterations than the subgradient method [Guignard and Rosenwein, 1989]; per iteration, the effort is of the same order. Hence, an ascent direction method is in general much faster than the subgradient method. However, it cannot be guaranteed to produce lower bounds that are as good. Ascent direction methods have shown to be successful for a wide range of combinatorial optimization problems. They include plant location problems [Bilde and Krarup, 1977; Erlenkotter, 1978; Guignard and Spielberg, 1979], the traveling salesman problem [Christofides, 1970; Balas and Christofides, 1981], the generalized assignment problem [Fisher, Jaikumar, and Van Wassenhove, 1985], and the set covering problem [Fisher and Kedia, 1990]. These applications indicate that the gain in speed over the subgradient method compensates the possible loss in lower bound quality more than sufficiently.

In spite of the numerous problems in machine scheduling, the use of ascent direction methods in this area has remained limited to two applications: Hariri and Potts [1984] and Potts [1985A] use ascent direction methods for the problems $F2|prec|C_{max}$ and $1|prec|\Sigma w_j C_j$, respectively. The problems were formulated in terms of 0-1 variables. The formulation of the latter problem is given in Section 1.3; the formulation of the former is similar. This thesis examines logical formulations for scheduling problems, such as the one we presented for the $1|prec|\Sigma w_j C_j$ problem in Section 1.3. In the subsequent chapters, we show that logical formulations facilitate the development of ascent direction methods.

In this subsection, we consider the ideas behind the design of ascent direction algorithms. We note that Guignard and Rosenwein [1989] present an application-oriented guide for designing Lagrangian ascent direction algorithms. Our discussion here is of a theoretical nature.

The notion of *directional derivative* plays a central role in ascent direction algorithms. The directional derivative of the function L at λ is defined as

$$L_u(\lambda) = \lim_{\epsilon \downarrow 0} \frac{L(\lambda + \epsilon u) - L(\lambda)}{\epsilon}, \tag{1.15}$$

for any vector $u \in \mathbb{R}^m$. Hence, λ is optimal if and only if

$$L_u(\lambda) \leqslant 0, \quad \text{for all } u \in \mathbb{R}^m; \tag{1.16}$$

note that this is equivalent to 0 being a subgradient at λ. If $L_{\bar{u}}(\lambda) > 0$ for some $\bar{u} \in \mathbb{R}^m$, then $\bar{u}$ is called an *ascent direction* of L at λ: we get an improved lower bound by moving some scalar step size Δ along $\bar{u}$. In general, it is difficult to compute directional derivatives. However, it is easy to compute them for the *primitive* vectors. A vector $u = (u_1, \ldots, u_m)$ is primitive if all $u_i = 0$ for all i but one. Hence, there are at most $2m$ different primitive directional derivatives at any λ.

We show that the primitive directional derivatives can almost be found by inspection. For $i = 1, \ldots, m$, let $l_i^+(\lambda)$ be the directional derivative at λ for any primitive vector with the ith component positive. For $i = 1, \ldots, m$, let $l_i^-(\lambda)$ be the directional derivative at λ for any primitive vector with the ith component negative. The first step is to rewrite definition (1.15) for these directional derivatives; accordingly, we get that

$$l_i^+(\lambda) = \lim_{\epsilon \downarrow 0} \frac{L(\lambda_1, \ldots, \lambda_i + \epsilon, \ldots, \lambda_m) - L(\lambda_1, \ldots, \lambda_m)}{\epsilon},$$

and

$$l_i^-(\lambda) = \lim_{\epsilon \downarrow 0} \frac{L(\lambda_1, \ldots, \lambda_i - \epsilon, \ldots, \lambda_m) - L(\lambda_1, \ldots, \lambda_m)}{\epsilon},$$

for $i = 1, \ldots, m$. Directional derivatives may not exist at the boundaries of the feasible region of λ; for instance, $l_i^-(\lambda)$ is undefined for $\lambda = (0, \ldots, 0)$, for any $i = 1, \ldots, m$.

The second step is to simplify the above expressions. First, recall that any problem (L_λ) may have multiple optimal solutions; let $X^{(\lambda)}$ be the set containing

them. Second, for any existing primitive direction at λ, a sufficiently small step size $\epsilon > 0$ exists such that some $x^+ \in X^{(\lambda)}$ remains optimal. This claim is proved as follows.

For any i ($i = 1, \ldots, m$), define $\lambda(\delta) = \lambda + \delta e^{(i)}$, where $e^{(i)}$ is the ith unity vector; accordingly, let $X^{(\lambda(\delta))}$ denote the set of optimal solutions for problem $(L_{\lambda(\delta)})$. Let $\delta(\bar{x}) = \{\delta \in \mathbb{R}^+ \cup \{0\} \mid \bar{x} \in X^{(\lambda(\delta))}\}$; $\delta(\bar{x})$ may be empty.

LEMMA 1.1. *$\delta(\bar{x})$ is a closed interval.*

PROOF. This is true because

$$\mu \in \delta(\bar{x}) \Leftrightarrow (c - \lambda(\delta)A)\bar{x} \leqslant (c - \lambda(\delta)A)x \quad \text{for all } x \in X.$$

(See also Figure 1.4). □

THEOREM 1.7. *There exists a sufficiently small value $\epsilon > 0$ such that some $x^+ \in X^{(\lambda)}$ is also optimal for problem* $(L_{\lambda(\epsilon)})$.

PROOF. For each $\bar{x} \in X$ and $\delta(\bar{x})$ not empty, let $\delta(\bar{x}) = [l(\bar{x}), r(\bar{x})]$. Determine

$$\epsilon = \tfrac{1}{2}\min\{\{l(\bar{x}) \mid l(\bar{x}) > 0, \bar{x} \in X^{(\lambda(\delta))}\}, 1\}.$$

Choose x^+ such that $\epsilon \in \delta(x^+)$; i.e., x^+ is an optimal solution for the problem $(L_{\lambda(\epsilon)})$. But this implies that $l(x^+) \leqslant 0$ and $r(x^+) \geqslant \epsilon$. □

Let now $\bar{\lambda} = (\lambda_1, \ldots, \lambda_i + \epsilon, \ldots, \lambda_m)$ for an arbitrary i ($i = 1, \ldots, m$) with $\epsilon > 0$, and let $x^+ \in X^{(\lambda)}$ also be optimal for problem $(L_{\bar{\lambda}})$ if ϵ is sufficiently small. Hence, x^+ must be such that

$$a^{(i)}x^+ = \max_{x \in X^{(\lambda)}} (a^{(i)}x),$$

where $a^{(i)}$ denotes the ith row of the matrix A. We get then that

$$L(\lambda_1, \ldots, \lambda_i + \epsilon, \ldots, \lambda_m) = L(\lambda_1, \ldots, \lambda_m) + \epsilon(b_i - a^{(i)}x^+),$$

if $\epsilon > 0$ and ϵ sufficiently small. Using this, we observe that the associated primitive direction reduces to

$$l_i^+(\lambda) = b_i - a^{(i)}x^+.$$

Note that

$$L(\bar{\lambda}) - L(\lambda) = \epsilon\, l_i^+(\lambda)$$

if ϵ is sufficiently small; hence, if $l_i^+(\lambda) > 0$, then we obtain an improved objective value by moving along the corresponding primitive direction. Similarly, we get

$$l_i^-(\lambda) = a^{(i)}x^- - b_i,$$

where x^- is such that

$$a^{(i)}x^- = \min_{x \in X^{(\lambda)}} (a^{(i)}x).$$

We now address the issue how to compute an appropriate step size to move by along an ascent direction. In general, we like to choose the step size so as to maximize the increment to the Lagrangian objective value. First of all, the step size must be feasible: it may not take us beyond the boundary of the feasible region of λ. Moving along a specified ascent direction but not crossing the boundary of the feasible region, we reach points where the directional derivative changes; possibly, we reach the point where the direction is no longer an ascent direction. Hence, any feasible step size that does not take us beyond this point induces an improved objective value. The step size maximizing the increment to the Lagrangian objective value either takes us to this point, or to a point at the boundary of the feasible region, whichever is closer. For instance, if $l_i^-(\lambda) > 0$, then the appropriate step size is found by maximizing

$$L(\lambda_1, \ldots, \lambda_i - \Delta, \ldots, \lambda_m)$$

subject to

$$0 \leqslant \Delta \leqslant \lambda_i.$$

Sometimes, other step sizes are preferable. Depending on the application, it may be more convenient to compute the step size that takes us to the first point where the directional derivative changes. We will see an example of this in Chapter 4.

The effectiveness of an ascent direction depends on our ability to specify an appropriate step size; the efficiency of an ascent direction algorithm depends on the effort to compute it. Apparently, we need to perform some kind of sensitivity analysis; the easier the Lagrangian problem, the easier it is to determine an appropriate step size.

In each iteration, we need to identify a primitive ascent direction, to specify an appropriate step size, and to adjust the vector of Lagrangian multipliers. This leaves freedom for implementation. For each application, we must determine in which order we scan the primitive directions, which one to choose (for instance, the direction of steepest ascent), which step size to compute, and so on.

An ascent direction method can of course be terminated if there is no significant increase of the objective value. Otherwise, termination occurs if no ascent direction among the existing primitive directions can be found anymore. Suppose an ascent direction method terminates at some $\bar{\lambda}$; if all primitive directional derivatives exist, then we have

$$l_i^+(\bar{\lambda}) \leqslant 0 \text{ and } l_i^-(\bar{\lambda}) \leqslant 0, \text{ for } i = 1, \ldots, m, \tag{1.17}$$

or, equivalently,

$$a^{(i)}x^- \leqslant b_i \leqslant a^{(i)}x^+, \text{ for } i = 1, \ldots, m.$$

We call these the *termination* conditions. Examining them, we see that they are necessary but not sufficient for optimality; hence, termination may occur having $\bar{\lambda} \neq \lambda^*$, i.e., before finding the optimal vector of Lagrangian multipliers. In addition, $\bar{\lambda}$ depends on the initial vector of Lagrangian multipliers, and on the ascent direction and the step size at each iteration. However, in view of the successful

applications we mentioned earlier, ascent direction methods are generally considered to be good approximation algorithms for problem (D).

In case $m = 1$, i.e., if only one constraint has been dualized, the conditions (1.17) are sufficient for optimality.

THEOREM 1.8. *An ascent direction algorithm solves problem* (D) *to optimality in a finite number of iterations if* $m = 1$.

PROOF. This follows immediately from the form of the function L for $m = 1$ (see Figure 1.4). Hopping from one point of non-differentiability to another in the direction of an optimal solution, we find an optimum in a finite number of iterations, since there is only a finite number of such points. □

For $m = 1$, binary search over the feasible interval of λ is an alternative approach to solve problem (D).

Many problems can be viewed as easy problems complicated by a single constraint. We mention the constrained shortest path problem [Handler and Zang, 1980] and the constrained linear assignment problem [Aggarwal, 1985]. For both problems, the Lagrangian problem possesses the integrality property, implying that the best Lagrangian lower bound equals the optimal solution of the linear programming relaxation (see Corollary 1.1). The latter can be obtained by use of a general linear programming algorithm; an ascent direction method, however, is much faster, since it employs the specialized algorithm for solving the Lagrangian problem. In Chapter 5, we address a scheduling problem with a single nasty constraint. Its Lagrangian dual problem is easy, and renders a lower bound that concurs with the upper bound for virtually all instances if the number of jobs is not too small.

It is often overlooked that the analysis of the conditions (1.17), which also apply to λ^*, may give valuable information about λ^* and the structure of an optimal dual solution. These conditions may be so forcing that λ^* can be partially or completely determined a priori. For instance, if some primitive directional derivative is non-negative everywhere, then the optimal value of the corresponding Lagrangian multiplier is attained at a boundary of the feasible region. In Chapters 2 and 6, we see examples of this phenomenon. Also in Chapter 2, we develop an approximation algorithm that is based upon the termination conditions (1.17).

Finally, we give here a sketch of an ascent direction method that illustrates the basic principles. We reconsider the generalized assignment problem, and analyze the Lagrangian problem obtained by dualizing the knapsack constraints. Since the Lagrangian problem is solvable in polynomial time, it is easy to develop an ascent direction method. Klastorin [1979] describes such a method for this problem; the ascent direction method we describe partly concurs with his.

Recall that in the solution to this Lagrangian problem, each job J_j is assigned to a machine M_i such that

$$c_{ij} + \lambda_i p_{ij} = \min_{1 \leq k \leq m} (c_{kj} + \lambda_k p_{kj}),$$

for $j = 1, \ldots, n$. From among the optimal solutions to the Lagrangian problem (L_λ), let $x(i)^+$ be the solution with the least jobs assigned to M_i, and let $x(i)^-$ be the solution with the most jobs assigned to M_i, for $i = 1, \ldots, m$. Furthermore, define $\mathcal{J}_i^+(\lambda)$ as the set of jobs assigned to M_i in $x(i)^+$, and define $\mathcal{J}_i^-(\lambda)$ as the set of jobs assigned to M_i in $x(i)^-$, for $i = 1, \ldots, m$. Hence, we have that

$$\mathcal{J}_i^+(\lambda) = \{J_j \mid c_{ij} + \lambda_i p_{ij} < \min_{1 \leqslant k \leqslant m, k \neq i}(c_{kj} + \lambda_k p_{kj})\},$$

for $i = 1, \ldots, m$; jobs with ties concerning the minimum dual cost $c_{ij} + \lambda_i p_{ij}$ have been assigned to other machines than M_i. Similarly, we have that

$$\mathcal{J}_i^-(\lambda) = \{J_j \mid c_{ij} + \lambda_i p_{ij} = \min_{1 \leqslant k \leqslant m}(c_{kj} + \lambda_k p_{kj})\},$$

for $i = 1, \ldots, m$; jobs with ties have been assigned to M_i. The primitive directional derivatives are then simply

$$l_i^+(\lambda) = \sum_{J_j \in \mathcal{J}_i^+(\lambda)} p_{ij} - b_i, \quad \text{for } i = 1, \ldots, m,$$

and

$$l_i^-(\lambda) = b_i - \sum_{J_j \in \mathcal{J}_i^-(\lambda)} p_{ij}, \quad \text{for } i = 1, \ldots, m.$$

Suppose now that $l_i^+(\lambda) > 0$; hence, the capacity b_i of M_i is exceeded, even if all ties have been settled in favor of other machines. Increasing λ_i, thereby making M_i more expensive, is an ascent direction. Moving along this direction by a specified step size Δ, we reach the first point where some job currently scheduled on M_i can equally well be scheduled on some other machine M_k; i.e., Δ is the smallest positive value for which there exists some job $J_j \in \mathcal{J}_i^+(\lambda)$ and some M_k such that

$$c_{ij} + (\lambda_i + \Delta) p_{ij} = c_{kj} + \lambda_k p_{kj};$$

hence, we have that

$$\Delta = -\lambda_i + \min_{1 \leqslant k \leqslant m,\, k \neq i,\, J_j \in \mathcal{J}_i^+(\lambda)} (c_{kj} + \lambda_k p_{kj} - c_{ij}) / p_{ij}.$$

At this point, the directional derivative changes. It is easy to verify that $L(\lambda_1, \ldots, \lambda_i + \Delta, \ldots, \lambda_m) - L(\lambda_1, \ldots, \lambda_i, \ldots, \lambda_m) = \Delta l_i^+(\lambda) > 0$. Suppose now that $l_i^-(\lambda) < 0$; M_i has spare capacity, even if jobs with ties have been assigned to M_i. Decreasing λ_i, thereby making M_i cheaper, is an ascent direction. Moving along this direction, we eventually reach the point where some J_j currently scheduled on $M_k \neq M_i$ is forced to go to M_i; i.e., the desired step size Δ is the smallest positive value for which there exists some J_j currently scheduled on M_k such that

$$c_{ij} + (\lambda_i - \Delta) p_{ij} = c_{kj} + \lambda_k p_{kj};$$

hence,

$$\Delta = \lambda_i - \min_{J_j \in \mathcal{J} - \mathcal{J}_i^-(\lambda), 1 \leqslant k \leqslant m, k \neq i} (c_{kj} + \lambda_k p_{kj} - c_{ij}) / p_{ij}.$$

Any ascent direction method built upon this principle terminates at some $\bar{\lambda}$ for

which

$$\sum_{J_j \in \mathcal{J}_i^+(\lambda)} p_{ij} \leqslant b_i \leqslant \sum_{J_j \in \mathcal{J}_i^-(\lambda)} p_{ij}, \qquad \text{for } i = 1, \ldots, m.$$

We note that Fisher, Jaikumar, and Van Wassenhove [1985] propose an ascent direction method for the alternative Lagrangian problem of the generalized assignment problem, which is obtained by dualizing the assignment constraints. Since this Lagrangian problem is only solvable in pseudo-polynomial time, the computation of the step size is difficult. Computational results exhibit, however, that the additional effort is worthwhile.

1.3.3.4. One-shot methods

A promising strategy, completely different from the iterative procedures we discussed, proceeds as follows. Recall that problem (D) can be represented as a linear program with a finite but huge number of constraints. The transition to a linear program proceeds by the representation of the finite set X as $X = \{x^{(1)}, \ldots, x^{(K)}\}$. Define now $\Lambda^{(k)}$ as

$$\Lambda^{(k)} = \{\lambda \geqslant 0 \mid (c - \lambda A)x^{(k)} + \lambda b = \min\{(c - \lambda A)x + \lambda b \mid x \in X\}\},$$

for $t = 1, \ldots, T$. Put in words, $\Lambda^{(k)}$ contains all the Lagrangian multipliers $\lambda \geqslant 0$ for which $x^{(k)} \in X$ solves the associated Lagrangian problem. Obviously, the optimal objective value of the following problem, referred to as problem $(\mathrm{S}^{(k)})$, which is to maximize

$$(c - \lambda A)x^{(k)} + \lambda b \qquad (\mathrm{S}^{(k)})$$

subject to

$$\lambda \in \Lambda^{(k)},$$

provides a lower bound on $L(\lambda^*)$, and therefore, on $v(\mathrm{P})$, for each k, $k = 1, \ldots, K$. We denote the optimal objective value of $(\mathrm{S}^{(k)})$ by $v(\mathrm{S}^{(k)})$. Note that $v(\mathrm{S}^{(k)}) = L(\lambda)$ for some $\lambda \in \Lambda^{(k)}$. It is not feasible to solve each of these problems; the strategy is therefore to single out some $x^{(k)}$ that can be expected to correspond to a comparatively strong lower bound $v(\mathrm{S}^{(k)})$; this is why we call it a *one-shot* method. An attractive choice for $x^{(k)}$ seems to be a feasible and good approximate solution for problem (P). The underlying logic comes from linear programming: if (P) is a linear program and if $x^{(k)}$ is its optimal solution, then we have $v(\mathrm{S}^{(k)}) = L(\lambda^*) = v(\mathrm{P})$. The whole idea is of course only feasible if problem $(\mathrm{S}^{(k)})$ displays some agreeable structure that makes it not too difficult to solve.

When stated formally, the problem $(\mathrm{S}^{(k)})$ has an intricate structure, mainly due to the requirement to specify $\Lambda^{(k)}$. For some single-machine scheduling problems, however, this requirement is easy to satisfy. Hariri and Potts [1983] and Potts and Van Wassenhove [1983,1985] exploit this for the problems $1|r_j|\Sigma w_j C_j$, $1|\bar{d}_j|\Sigma w_j C_j$, and $1||\Sigma w_j T_j$, respectively. The first two problems are formulated in terms of the job completion times $C_1, \ldots, C_n$, similar to the formulation we gave for $1|prec|\Sigma w_j C_j$. The total weighted tardiness problem is

formulated similarly in terms of both the job completion times and the job tardinesses $T_1, \ldots, T_n$. After dualizing the proper constraints, determining a primal feasible sequence σ, computing the corresponding job completion times $\bar{C}_1, \ldots, \bar{C}_n$, and reindexing the jobs in order of appearance, each problem $(S^{(k)})$ for each of the applications is basically of the following type: maximize

$$\sum_{j=1}^{n} \lambda_j \bar{C}_j - \sum_{j=1}^{n} \lambda_j a_j$$

subject to

$$\lambda_1/p_1 \geqslant \lambda_2/p_2 \geqslant \cdots \geqslant \lambda_n/p_n, \tag{1.18}$$

$$\lambda_j \geqslant 0, \quad \text{for } j = 1, \ldots, n, \tag{1.19}$$

where a_j $(j = 1, \ldots, n)$ is a non-negative integer that depends on the parameters of the problem. After all, the prespecified sequence σ is optimal for this problem only if the jobs are sequenced in compliance with Smith's rule; this is expressed by the conditions (1.18). This type of problem is solvable in $O(n \log n)$ time, and has been empirically shown to provide satisfactorily strong lower bounds for the respective applications. For the $1||\Sigma w_j T_j$ problem in particular, this lower bound approach has led to an efficient branch-and-bound algorithm. For the machine scheduling problems dealt with in Chapters 2, 4, and 5, we work in a reverse manner: we present there empirical evidence that the $x^{(k)}$ that are good approximate solutions for the Lagrangian dual problem (D) are also good approximate solutions for the primal problem (P).

1.3.3.5. Miscellaneous approaches

Since any $\lambda \geqslant 0$ induces a lower bound, many approaches for approximating the optimal solution of problem (D) are conceivable. Many of these are hybrids, often with some flavor of the subgradient method and of an ascent direction method. Sometimes, different approximation algorithms for problem (D) are used at the different levels of the search tree. For instance, for the $R||C_{\max}$ problem in Chapter 4, we find a good approximation for λ^* and use this approximation throughout the search tree with some minor adjustments. Many variants of this concept exist. In general, they are empirically motivated.

1.3.4. Competing relaxations and formulations

As the trade-off between lower bound quality and the time needed to compute it is the most important issue involved in the application of Lagrangian relaxation, we must be prepared to consider a variety of relaxations. We may have to analyze different relaxations coming from the same formulation (cf. the generalized assignment problem), but also relaxations that come from different formulations of the problem (cf. the $1|prec|\Sigma w_j C_j$ problem). Furthermore, a problem may not only have multiple integer linear programming formulations, but it may also permit completely different formulations.

How to choose between competing formulations and relaxations? Often, it can only be established empirically what the most attractive relaxation is. Quality-

wise, we have seen that the lower bounds from different relaxations from the same formulation can be compared analytically under certain circumstances. In a recent paper, Dyer and Wolsey [1990] even manage to establish a hierarchy of relaxations for the $1|r_j|\Sigma w_jC_j$ problem that stem from different formulations. Yet, the quality of the lower bound is not the decisive factor in the choice of a relaxation; the time needed to compute it should also be taken into consideration. As a coarse guideline, Lagrangian problems with few constraints dualized seem to be more attractive; on the one hand, fewer Lagrangian multipliers are involved, on the other hand, the Lagrangian problem bears a stronger resemblance to the original problem.

In addition to the efficiency and effectiveness of the lower bound, other issues may play a role, too. Relaxations may give rise to elimination criteria, by which some feasible subsets can be excluded from further consideration. For the $R||C_{max}$ problem, for instance, it is possible to reduce the problem size by fixing some of the assignment variables as a result of the lower bound (see Chapter 4). For $1|prec|\Sigma w_jC_j$, it is possible to derive additional precedence constraints between the jobs (see Chapter 2). Another important feature is the extent to which Lagrangian problems induce approximate solutions. Sometimes, the solution to the Lagrangian problem is also feasible for the primal problem, or it may suffice to perturb the solution slightly to obtain one. Solving the Lagrangian problem of the generalized assignment problem obtained by dualizing the knapsack constraint, we get for each vector λ an assignment of jobs to machines. Perhaps this assignment is already feasible, perhaps only minor adjustments suffice to make it feasible. In the subsequent chapters, we show examples of this, and provide empirical evidence that good approximate solutions for scheduling problems can be obtained in this manner.

1.4. Machine scheduling and Lagrangian relaxation

1.4.1. A review of the literature

Lagrangian relaxation has been intensively analyzed for and successfully applied to a wide variety of combinatorial optimization problems (see Fisher [1985] and Guignard and Rosenwein [1989]). This assertion can be challenged, however, as far as machine scheduling problems are concerned. The analysis of Lagrangian relaxation and its application form a relatively small portion of the voluminous research, including branch-and-bound algorithms, in this area.

A likely reason for this is the inclination to formulate machine scheduling problems as integer linear programs. Integer linear programming formulations usually suffer severely from a high number of variables or constraints. It is not so surprising then that they seldom give rise to promising relaxations. The inclination towards integer linear programming formulations may prohibit the examination of other formulations, in which attractive underlying structures may become apparent. By 'other formulations', we specifically refer to formulations in terms of the job completion times and disjunctive constraints like the one for $1|prec|\Sigma w_jC_j$ in Section 1.3.

A review of the literature on papers involving Lagrangian relaxation applied to

machine scheduling can be brief and reverts mostly to the references given in Section 1.3. If we make a distinction between integer linear programming formulations and other formulations, then the following papers fall in the first category. Fisher [1973] explores the possibilities of Lagrangian relaxation for machine scheduling problems in a broad context, and was probably the first to do so. We mention Hariri and Potts [1984] and Potts [1985A], who consider the problems $F2|prec|C_{max}$ and $1|prec|\Sigma w_j C_j$, respectively. The formulation for $1|prec|\Sigma w_j C_j$ employed by Potts is given in Section 1.3. The formulation for $F2|prec|C_{max}$ is also based upon variables x_{jk} that take the value 1 if J_j is sequenced before J_k and the value 0 otherwise. For both applications, ascent direction methods are developed.

For applications that fall in the second category, we refer to Hariri and Potts [1983] and Potts and Van Wassenhove [1983,1985] for the problems $1|r_j|\Sigma w_j C_j$, $1|\bar{d}_j|\Sigma w_j C_j$, and $1||\Sigma w_j T_j$, respectively. Lower bounds for these problems are obtained by the one-shot methods we discussed in Section 1.3.3.4. Furthermore, Fisher [1976] presents a formulation in terms of the job start times for the $1||\Sigma T_j$ problem, but uses a pseudo-polynomial number of constraints to make sure that the machine handles no more than one job at a time. The subgradient method is applied to find strong lower bounds.

1.4.2. A preview of this thesis

The following chapters deal with the application of Lagrangian relaxation and Lagrangian duality to a variety of machine scheduling problems. These include single-machine, flow-shop, and parallel-machine problems, and involve assignment, partitioning, sequencing, and scheduling problems. We particularly examine logical formulations that facilitate the development of ascent direction methods. The application of Lagrangian relaxation is not only focused on lower bounds, but also on upper bounds. In general, Lagrangian relaxation and duality are problem-specific, and provide additional opportunities beyond lower bound and upper bound computations. In these cases, we give full descriptions of the branch-and-bound algorithms. The design of these branch-and-bound algorithms is usually steered by the analysis of the Lagrangian problems.

In Chapter 2, however, we immediately deviate from this course, since the analysis here is not problem-specific, but problem-class-specific. We consider Lagrangian relaxation and Lagrangian duality for single-machine problems that can be formulated in terms of the job completion times. For such a problem, it is relatively simple to design an ascent direction method. Upon termination of the ascent direction method, we obtain a decomposition of the problem, which we call *dual decomposition.* We try to verify to what extent the dual decomposition concurs with a correct decomposition. The dual decomposition serves as a framework or as a guide for the design of approximative and enumerative methods. We demonstrate this by a detailed analysis of the $1|prec|\Sigma w_j C_j$ problem. This chapter is based on Van de Velde [1990B].

In Chapter 3, we consider the problem of minimizing the sum of the job completion times in the 2-machine flow-shop. This problem is formulated in terms of the job completion times on both machines. It is not so surprising that the

Lagrangian problem decomposes initially into two single-machine problems. However, it reduces to a linear ordering problem when a restriction is added that is redundant for the primal problem. Although the linear ordering problem is $\mathcal{NP}$-hard, it turns out to be well-solvable for certain choices of the Lagrangian multipliers. It is also shown how this approach can be extended to the general case of m machines. This chapter draws upon from Van de Velde [1990A].

In Chapter 4, we analyze the problem of minimizing the makespan on unrelated parallel machines. This problem has already been introduced in Section 1.1.3. We give an integer linear programming formulation, and dualize a set of constraints by which we obtain a Lagrangian problem with the integrality property. We show, nonetheless, that the Lagrangian problem provides a solid basis for the development of an optimization algorithm and an approximation algorithm. Chapter 4 is closely related to work presented in Van de Velde [1990C].

In Chapter 5, we consider a single-machine problem where the jobs have a common due date. The objective is to minimize the sum of deviations of the job completion times from the common due date. The machine scheduling problems in which early completions of jobs are penalized as well form a rapidly evolving field of research, which is inspired by the just-in-time concept for manufacturing. Much concurrent research concerns problems of this type, and this problem in particular. When formulated as an easy partitioning problem complicated by a single constraint, the success of Lagrangian relaxation and duality is remarkable: the Lagrangian dual problem gives rise to a lower and an upper bound that concur for virtually all instances with the number of jobs not too small. Chapter 5 is based upon Hoogeveen, Oosterhout, and Van de Velde [1990].

In Chapter 6, we examine the single-machine scheduling problem of minimizing total inventory cost. This objective is reflected by a linear combination of two cost functions, one of which penalizes early completions. Apart from the objective function, this problem differs on two counts from the problem examined in Chapter 5. The due dates may be distinct, and it may be profitable to insert machine idle time between the execution of jobs. The latter is the intriguing aspect of this problem. It may seem to be a scheduling problem at first sight. Yet, there is a one-to-one correspondence between sequencing and scheduling, as there is a polynomial method to optimally insert machine idle time for a given sequence. Hence, the problem reduces to a sequencing problem after all. Very little is known about the derivation of lower bounds and the development of branch-and-bound algorithms for such problems. The possible occurrence of machine idle time complicates the matter of scheduling significantly, since this aspect is hard to capture in lower bound methods of any nature. We show that even for such problems Lagrangian relaxation is a useful method. We present a number of other lower bound strategies and elimination criteria for this problem as well, as it appears that Lagrangian relaxation is a less dominant technique here than it is for the previous problems. Chapter 6 is based upon Hoogeveen and Van de Velde [1991B].

2

Single-Machine Scheduling

We present in this chapter a framework for the analysis of Lagrangian problems associated with single-machine scheduling problems that can be formulated as follows: determine job completion times $C_1, \ldots, C_n$ that minimize

$$f(C_1, \ldots, C_n, T_1, \ldots, T_n, E_1, \ldots, E_n)$$

subject to

$$g_h(C_1, \ldots, C_n, T_1, \ldots, T_n, E_1, \ldots, E_n) \leqslant 0, \text{ for } h = 1, \ldots, H, \quad (2.1)$$

the capacity and availability of the machine, (2.2)

where f and g_h $(h = 1, \ldots, H)$ are functions that are linear in their arguments. T_j is the tardiness of J_j, defined as $T_j = \max\{C_j - d_j, 0\}$, and E_j is the earliness of J_j, defined as $E_j = \max\{d_j - C_j, 0\}$. In addition, we assume that H is bounded by a polynomial in n, the number of jobs.

The conditions (2.2) generally state that the machine can handle no more than one job at a time and that it is continuously available only from time 0 onwards. The conditions (2.1) are assumed to be the nasty constraints: if they are absent, then the problem is solvable in polynomial time.

The formulation covers many single-machine scheduling problems, including minimizing $\Sigma_{j=1}^{n} w_j C_j$ subject to precedence constraints and minimizing total weighted tardiness. These problems are considered in this chapter. Such a formulation followed by the Lagrangian relaxation of the type (2.1) constraints is also used by Hariri and Potts [1983] for the problem of minimizing total weighted completion time subject to release times, by Potts and Van Wassenhove [1984] for minimizing total weighted completion time subject to deadlines, and by Potts and Van Wassenhove [1985] for minimizing total weighted tardiness. These authors use application-specific one-shot methods to approximate the optimal

solution value of the Lagrangian dual problem (see Section 1.3.3.4).

The Lagrangian problem, obtained by dualizing the constraints (2.1), offers attractive opportunities for the development of both optimization and approximation algorithms. We demonstrate this by the analysis of $1|prec|\Sigma w_j C_j$ (i.e., the problem of minimizing total weighted completion time subject to precedence constraints). The analysis is typical of single-machine scheduling problems that allow the generic formulation.

This chapter is organized as follows. In Section 2.1, we introduce the $1|prec|\Sigma w_j C_j$ problem, formulate it in terms of the job completion times, derive the Lagrangian problem, develop a quick ascent direction method, and analyze the termination conditions. Upon termination of the ascent direction method, we get a decomposition of the jobs into subsets; we call this a *dual decomposition*. In Section 2.2, we show how such a dual decomposition can be employed to find approximate solutions for the primal problem. Computational results exhibit the high quality of such solutions. In Section 2.3, we discuss how the dual decomposition can be of use for the design of enumerative methods. In Section 2.4, we show that a similar analysis can be conducted for the problem of minimizing total weighted tardiness.

2.1. The Lagrangian dual of $1|prec|\Sigma w_j C_j$

We consider the following single-machine problem. A set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n independent jobs has to be scheduled on a single machine that can handle no more that one job at a time. The machine is continuously available from time 0 onwards. Each job J_j ($j = 1, \ldots, n$) requires processing during an uninterrupted period of a given length p_j. In addition, each job J_j has a weight w_j, expressing its urgency relative to other jobs. We assume that the processing times and weights are integral. There are precedence constraints present between the jobs. They are represented by an acyclic precedence graph G with vertex set $\{J_1, \ldots, J_n\}$ and arc set A, which equals its transitive reduction. A path in G from J_j to J_k implies that J_j has to be executed before J_k; J_j is a *predecessor* of J_k, and J_k is a *successor* of J_j. In case there is an arc $(J_j, J_k) \in A$, then J_j is said to be an *immediate predecessor* of J_k; J_k is then an *immediate successor* of J_j. We define $\mathcal{P}_j$ and $\mathcal{S}_j$ as the set of immediate predecessors and immediate successors of J_j, respectively ($j = 1, \ldots, n$). A *schedule* is a specification of the job completion times, denoted by C_j ($j = 1, \ldots, n$), such that the jobs do not overlap in their execution. The objective is to find a feasible schedule that minimizes the total weighted completion time, $\Sigma_{j=1}^n w_j C_j$.

If there are no precedence constraints, then the problem is solvable by Smith's ratio rule [Smith, 1959]: simply process the jobs in order of non-increasing values w_j/p_j in the interval $[0, \Sigma_{j=1}^n p_j]$ (see Theorem 1.1). For special classes of precedence constraints, the problem is still solvable in $O(n \log n)$ time; this is the case for tree-like precedence constraints [Horn, 1972; Adolphson and Hu, 1973; Sidney, 1975] and for series-parallel precedence constraints [Lawler, 1978]. In general, the problem is $\mathcal{NP}$-hard in the strong sense [Lawler, 1978; Lenstra and Rinnooy Kan, 1978]. This justifies the development of approximative and enumerative algorithms. Morton and Dharan [1978] propose several heuristics.

Specifically, the so-called *tree-optimal-heuristic*, producing optimal solutions in case the precedence constraints take the form of a tree, generates high-quality solutions. Potts [1985A] presents a branch-and-bound that solves instances up to 100 jobs; he employs Lagrangian lower bounds obtained from a 0-1 linear programming formulation of the problem.

In contrast to Potts [1985A], we formulate the $1|prec|\Sigma w_j C_j$ problem in terms of the job completion times. For a comparison of the two formulations, we refer to Section 1.3. The problem is then formulated as follows: determine job completion times that minimize

$$\sum_{j=1}^{n} w_j C_j$$

subject to

$$C_k \geqslant C_j + p_k, \quad \text{for each } (J_j,J_k) \in A, \tag{2.3}$$

$$C_j \geqslant C_k + p_j \text{ or } C_k \geqslant C_j + p_k, \quad \text{for } j, k = 1, \ldots, n, \ j \neq k, \tag{2.4}$$

$$C_j - p_j \geqslant 0, \quad \text{for } j = 1, \ldots, n. \tag{2.5}$$

The conditions (2.3) stipulate the precedence constraints; the conditions (2.4) and (2.5) reflect the traditional assumptions regarding machine capacity and availability.

The $1|prec|\Sigma w_j C_j$ problem can be seen as an easy-to-solve problem complicated by the conditions (2.3). Accordingly, we introduce a vector $\lambda \in \mathbb{R}^A$ that contains a Lagrangian multiplier $\lambda_{jk} \geqslant 0$ for each arc $(J_j,J_k) \in A$ and put the constraints (2.3), each weighted by its multiplier, into the objective function. For a given vector $\lambda \geqslant 0$, the Lagrangian relaxation problem, referred to as problem (L_λ), is to find $L(\lambda)$, which is the minimum of

$$\sum_{j=1}^{n} \left[(w_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} - \sum_{J_k \in \mathcal{P}_j} \lambda_{kj}) C_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} p_k \right] \tag{L_λ}$$

subject to the machine capacity and availability conditions (2.4) and (2.5).

For $j = 1, \ldots, n$, let $w_j'(\lambda) = (w_j + \Sigma_{J_k \in \mathcal{S}_j} \lambda_{jk} - \Sigma_{J_k \in \mathcal{P}_j} \lambda_{kj})/p_j$; we call $w_j'(\lambda)$ the *relative weight* of job J_j. Using Smith's ratio rule, we solve problem (L_λ) by sequencing the jobs in order of non-increasing relative weights. For any $\lambda \geqslant 0$, $L(\lambda)$ is a lower bound for the $1|prec|\Sigma w_j C_j$ problem; we like therefore to find the vector λ^* that induces the best Lagrangian lower bound. This is the Lagrangian dual problem, referred to as problem (D): maximize

$$L(\lambda) \tag{D}$$

subject to

$$\lambda_{jk} \geqslant 0, \quad \text{for each } (J_j,J_k) \in A.$$

Following the lines of Section 1.3.3, we first prove that problem (D) is solvable to optimality in polynomial time. For problem (D), at most $n!$ feasible sequences are involved; the tth feasible sequence induces the vector $(C_1^{(t)}, C_2^{(t)}, \ldots, C_n^{(t)})$ of

job completion times, where $t = 1, \ldots, T$ and $T \leqslant n!$. Problem (D) is then equivalent to the following problem: maximize

$$z$$

subject to

$$z \leqslant \sum_{j=1}^{n} \left[(w_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} - \sum_{J_k \in \mathcal{P}_j} \lambda_{kj}) C_j^{(t)} + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} p_k \right], \text{ for } t = 1, \ldots, T,$$

$$\lambda_{jk} \geqslant 0, \qquad \text{for each } (J_j, J_k) \in A.$$

Problem (D) has been transformed into a problem of maximizing a linear function subject to a finite number of linear constraints.

THEOREM 2.1. *Problem* (D) *is solvable in polynomial time through Khachiyan's ellipsoid method* [Khachiyan, 1979].

PROOF. The proof proceeds in the same spirit as the proof of Theorem 1.6; it is included for sake of completeness. Let

$$\mathcal{K} = \{(z, \lambda) \mid z \in \mathbb{R}, \lambda \times \mathbb{R}^A, z \leqslant \sum_{j=1}^{n} \left[(w_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} - \sum_{J_k \in \mathcal{P}_j} \lambda_{kj}) C_j^{(t)} + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} p_k \right],$$

$$\text{for } t = 1, \ldots, T,\ \lambda_{jk} \geqslant 0, \text{ for each } (J_j, J_k) \in A\ \}.$$

Recall that it suffices to show that the following *separation problem for* $\mathcal{K}$ is solvable in polynomial time (see Grötschel, Lovász, and Schrijver [1981], and Padberg and Rao [1982]): given $(\bar{z}, \bar{\lambda}) \in \mathbb{Q} \times \mathbb{Q}^A$, decide whether $(\bar{z}, \bar{\lambda}) \in \mathcal{K}$; if not, give a *separating hyperplane*, that is, an inequality $\lambda d^{\mathrm{T}} + \alpha z \leqslant \beta$, such that

$$(z, \lambda) \in \mathcal{K} \Rightarrow \lambda d^{\mathrm{T}} + \alpha z \leqslant \beta,$$

and

$$\bar{\lambda} d^{\mathrm{T}} + \alpha \bar{z} > \beta.$$

Observe now that for a given $(\bar{z}, \bar{\lambda})$ we determine the value $L(\bar{\lambda})$ and the corresponding vector $(\bar{C}_1, \bar{C}_2, \ldots, \bar{C}_n)$ of job completion times by solving problem $(\mathrm{L}_{\bar{\lambda}})$; this is done in $O(n \log n)$ time. If $L(\bar{\lambda}) \geqslant \bar{z}$, then $(\bar{z}, \bar{\lambda}) \in \mathcal{K}$; if $L(\bar{\lambda}) < \bar{z}$, then $(\bar{z}, \bar{\lambda}) \notin \mathcal{K}$, and

$$\bar{z} > \sum_{j=1}^{n} \left[(w_j + \sum_{J_k \in \mathcal{S}_j} \bar{\lambda}_{jk} - \sum_{J_k \in \mathcal{P}_j} \bar{\lambda}_{kj}) \bar{C}_j + \sum_{J_k \in \mathcal{S}_j} \bar{\lambda}_{jk} p_k \right],$$

is a separating hyperplane. □

In practice, the ellipsoid method is very slow; we develop therefore a quick ascent direction algorithm to approximate the optimal solution of problem (D). First, we derive expressions for the primitive directional derivatives. In an

optimal solution for the Lagrangian problem, the position of J_j depends on its relative weight: the larger its relative weight, the smaller its completion time. If its weight is tied, then its position also depends on the way ties are settled. Let now $C_j^+(\lambda)$ denote the earliest possible completion time of J_j in an optimal schedule for problem (L_λ); let $C_j^-(\lambda)$ denote the latest possible completion time of J_j in an optimal schedule for problem (L_λ). Increasing λ_{jk} by a specific $\epsilon > 0$ will increase the relative weight of J_j from $w_j'(\lambda)$ to $w_j'(\lambda) + \epsilon/p_j$; simultaneously, it will decrease the relative weight of J_k from $w_k'(\lambda)$ to $w_k'(\lambda) - \epsilon/p_k$. It is possible to choose $\epsilon > 0$ small enough to ensure that at least one optimal schedule for problem (L_λ) remains optimal (see Theorem 1.7). In such an optimal schedule, J_j must be completed on time $C_j^+(\lambda)$ and J_k must be completed on time $C_k^-(\lambda)$. Increasing λ_{jk} by such a small ϵ affects the Lagrangian objective value by $\epsilon(C_j^+(\lambda) - C_k^-(\lambda) + p_k)$. From this, we derive that the primitive directional derivative for increasing λ_{jk} at λ, denoted by $l_{jk}^+(\lambda)$, is

$$l_{jk}^+(\lambda) = C_j^+(\lambda) - C_k^-(\lambda) + p_k, \quad \text{for each } (J_j, J_k) \in A.$$

If $l_{jk}^+(\lambda) > 0$, then increasing λ_{jk} is an ascent direction: we get an improved objective value by moving along this direction. The sign of each $l_{jk}^+(\lambda)$ is determined in constant time. Note that for each arc $(J_j, J_k) \in A$, we have

$$C_j^+(\lambda) > C_k^-(\lambda) + p_k \Leftrightarrow w_j'(\lambda) < w_k'(\lambda);$$

hence, $l_{jk}^+(\lambda) > 0 \Leftrightarrow w_j'(\lambda) < w_k'(\lambda)$. In a similar fashion, we find that $l_{jk}^-(\lambda)$, the primitive directional derivative for decreasing λ_{jk} at each λ with $\lambda_{jk} > 0$, is

$$l_{jk}^-(\lambda) = C_k^+(\lambda) - C_j^-(\lambda) - p_k, \quad \text{for each } (J_j, J_k) \in A.$$

If $l_{jk}^-(\lambda) > 0$, then decreasing λ_{jk} is an ascent direction: we get an improved objective value by moving along this direction.

Given an ascent direction, we invariably move by the step size that maximizes the increment to the objective value. If $l_{jk}^+(\lambda) > 0$, then the increment is maximized by moving to the first point where increasing λ_{jk} is no longer an ascent direction. At this point, the relative weights of J_j and J_k are equal. Hence, the required step size is the value Δ for which

$$w_j'(\lambda) + \Delta/p_j = w_k'(\lambda) - \Delta/p_k;$$

it is determined in constant time. Consider now the case $l_{jk}^-(\lambda) > 0$. To ensure that the Lagrangian vector remains non-negative, we impose the condition that $\Delta \leqslant \lambda_j$. If this condition is not restrictive, then we move to the first point where decreasing λ_j is no longer an ascent direction. If it is restrictive, then we take the step size as large as possible. Hence, the step size that maximizes the increment of the objective value is computed as the largest value $\Delta \leqslant \lambda_j$ for which

$$w_j'(\lambda) - \Delta/p_j \geqslant w_k'(\lambda) + \Delta/p_k.$$

Eventually, termination occurs at some $\bar{\lambda}$ at which no ascent direction exists anymore. Later on, we will analyze the termination conditions. We first give a step-wise description of the ascent direction algorithm.

ASCENT DIRECTION ALGORITHM

Step 0. Set $\lambda_{jk} = 0$ for each $(J_j, J_k) \in A$, and compute the relative weights $w_j'(\lambda)$.
Step 1. For each $(J_j, J_k) \in A$, do the following:
(a) If $w_j'(\lambda) > w_k'(\lambda)$, then compute the step size

$$\Delta = [w_k'(\lambda) - w_j'(\lambda)]p_j p_k / (p_j + p_k).$$

Put $\lambda_{jk} \leftarrow \lambda_{jk} + \Delta$, and update $w_j'(\lambda)$ and $w_k'(\lambda)$.
(b) If $w_j'(\lambda) < w_k'(\lambda)$, then compute the step size

$$\Delta = \max\{\lambda_j, [w_j'(\lambda) - w_k'(\lambda)]p_j p_k / (p_j + p_k)\}.$$

Put $\lambda_{jk} \leftarrow \lambda_{jk} - \Delta$, and update $w_j'(\lambda)$ and $w_k'(\lambda)$.
Step 2. If no multiplier adjustment has taken place, then compute $L(\lambda)$ and stop; otherwise, return to Step 1.

Let I be the number of times that Step 1 is executed. The ascent direction algorithm runs then in $O(I|A| + n\log n)$ time. Since we cannot bound I by a polynomial in n and $|A|$, the ascent direction algorithm is presumably not a polynomial-time method. In practice, however, the algorithm is very fast, producing very good approximate solutions.

THEOREM 2.2. *The ascent direction algorithm described above generates a series of monotonically increasing lower bounds for problem* (P).

PROOF. Given an arbitrary $\lambda \geq 0$, we first assume $w_j'(\lambda) < w_k'(\lambda)$; hence, $l_{jk}^+(\lambda) = C_j^+(\lambda) - C_k^-(\lambda) + p_k > 0$, and increasing λ_{jk} is an ascent direction. We reindex the jobs according to non-increasing relative weights, settling all ties arbitrarily except for J_j and J_k: we give J_j the smallest index possible and J_k the largest index possible. Let $C_1, \ldots, C_n$ be the job completion times for the sequence $(J_1, \ldots, J_n)$; note that $C_j = C_j^+(\lambda)$ and $C_k = C_k^-(\lambda)$. Hence, in more detail, the schedule under consideration is $(J_1, \ldots, J_{k-1}, J_k, J_{k+1}, \ldots, J_{j-1}, J_j, J_{j+1}, \ldots, J_n)$. Let Δ be the step size as prescribed, and let $\bar{\lambda}$ denote the vector of Lagrangian multipliers after increase of λ_{jk} by Δ. Since λ and $\bar{\lambda}$ differ only in one component, the relative weights for all jobs but J_j and J_k remain the same. An optimal schedule for problem $(L_{\bar{\lambda}})$ is then $(J_1, \ldots, J_{k-1}, J_{k+1}, \ldots, J_l, J_j, J_k, J_{l+1}, \ldots, J_{j-1}, J_{j+1}, \ldots, J_n)$, for some J_l with $k+1 \leqslant l \leqslant j-1$; the job completion times for this schedule can conveniently be expressed in terms of $C_1, \ldots, C_n$. We now demonstrate that $L(\bar{\lambda}) > L(\lambda)$; it is basically a matter of writing out. For brevity, we let $\mu_i = w_i + \Sigma_{J_h \in \mathcal{S}_i} \lambda_{ih} - \Sigma_{J_h \in \mathcal{P}_i} \lambda_{hi}$ for each i $(i = 1, \ldots, n)$. We have

$$L(\bar{\lambda}) = \sum_{i=1}^{k-1} \mu_i C_i + \sum_{i=j+1}^{n} \mu_i C_i + \sum_{i=k+1}^{l} \mu_i (C_i - p_k) + \sum_{i=l+1}^{j-1} \mu_i (C_i + p_j) +$$
$$(\mu_k - \Delta)\left[C_k^-(\lambda) + p_j + \sum_{i=k+1}^{l} p_i\right] + (\mu_j + \Delta)\left[C_j^+(\lambda) - p_k - \sum_{i=l+1}^{j-1} p_i\right] +$$

$$\sum_{i=1}^{n} \sum_{J_h \in S_i} \lambda_{ih} p_h + \Delta p_k$$

$$= L(\lambda) + \sum_{i=l+1}^{j-1} (\mu_i p_j - \mu_j p_i) + \sum_{i=k+1}^{l} (\mu_k p_i - \mu_i p_k) + \mu_k p_j - \mu_j p_k +$$

$$\Delta \left[(C_j^+(\lambda) - p_k - \sum_{i=l+1}^{j-1} p_i) - (C_k^-(\lambda) + p_j + \sum_{i=k+1}^{l} p_i) \right] + \Delta p_k.$$

Since J_j and J_k are adjacent in the second schedule, we have that

$$(C_j^+(\lambda) - p_k - \sum_{i=l+1}^{j-1} p_i) - (C_k^-(\lambda) + p_j + \sum_{i=k+1}^{l} p_i) = -p_k.$$

This implies that

$$L(\bar{\lambda}) = L(\lambda) + \sum_{i=l+1}^{j-1} (\mu_i p_j - \mu_j p_i) + \sum_{i=k+1}^{l} (\mu_k p_i - \mu_i p_k) + \mu_k p_j - \mu_j p_k$$

$$= L(\lambda) + \sum_{i=k+1}^{l} (\mu_k / p_k - \mu_i / p_i) p_i p_k + \sum_{i=l+1}^{j-1} (\mu_i / p_i - \mu_j / p_j) p_i p_j +$$

$$(\mu_k / p_k - \mu_j / p_j) p_j p_k.$$

$$= L(\lambda) + \sum_{i=k+1}^{l} [w_k'(\lambda) - w_i'(\lambda)] p_i p_j + \sum_{i=l+1}^{j-1} [w_i'(\lambda) - w_j'(\lambda)] p_i p_j +$$

$$[w_k'(\lambda) - w_j'(\lambda)] p_j p_k.$$

Since $w_k'(\lambda) > w_j'(\lambda)$, $w_i'(\lambda) < w_k'(\lambda)$ for each i ($i = k+1, \ldots, l$), and $w_i'(\lambda) > w_j'(\lambda)$ for each i ($i = l+1, \ldots, j-1$), we find that $L(\bar{\lambda}) > L(\lambda)$.

The analysis for the case $l_{jk}^-(\lambda) > 0$ proceeds in a similar fashion. □

Consider the 10-job example from Potts [1985A] for which the processing times, weights, and precedence graph are given in Table 2.1 and Figure 2.1. If we put $\lambda_{jk} = 0$ for all $(J_j, J_k) \in A$, then an optimal schedule is $(J_3, J_{10}, J_4, J_9, J_7, J_6, J_2, J_5, J_8, J_1)$ with total cost 1055. The same schedule and lower bound are obtained by disregarding the precedence constraints and solving $1||\Sigma w_j C_j$. The schedule is not feasible for the original problem; for instance, J_{10} is executed before J_6 although $(J_6, J_{10}) \in A$. Since $w_6'(\lambda) < w_{10}'(\lambda)$, increasing $\lambda_{6,10}$ is an ascent direction. The appropriate step size is $\Delta = 17/7$, giving $\lambda_{6,10} = 17/7$. We get $(J_3, J_4, J_9, J_7, J_6, J_{10}, J_2, J_5, J_8, J_1)$ as an optimal schedule for the new Lagrangian problem, having value $L(\lambda) = 1106$. Proceeding along these lines, we get the value $L(\lambda) = 1526.69$ upon termination. Potts' procedure, requiring $\Omega(n^4)$ time, produces the lower bound 1519; the upper bound generated by the tree-optimal-heuristic is 1530. The duality gap is therefore no more than 3.31.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
p_j	6	9	1	3	9	5	7	7	6	2
w_j	2	5	9	6	5	4	9	3	8	5

TABLE 2.1. Processing times and weights.

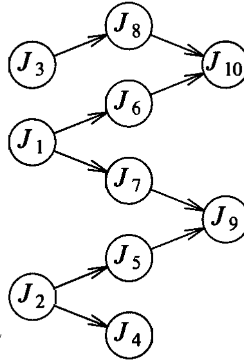


FIGURE 2.1. Precedence graph.

In the remainder of this chapter, we let $\bar{\lambda}$ denote the vector of Lagrangian multipliers upon termination of the ascent direction method. Using the termination conditions that all primitive directional derivatives are non-positive, we derive some properties for $\bar{\lambda}$ and for the optimal solutions of problem ($L_{\bar{\lambda}}$). These properties are important for the development of approximation and optimization algorithms for $1|prec|\Sigma w_j C_j$.

DEFINITION 2.1. The job set $\mathcal{B} \subseteq \mathcal{J}$ is called a *block* for a given $\lambda \geqslant 0$ if

$$(w_j + \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} - \sum_{J_k \in \mathcal{P}_j} \lambda_{kj})/p_j = c, \quad \text{for each } J_j \in \mathcal{B},$$

where c is some positive real constant.

In any optimal schedule to problem (L_λ), the jobs in a block are interchangeable without affecting the Lagrangian objective value $L(\lambda)$. For any given $\lambda \geqslant 0$, the job set $\mathcal{J}$ is decomposed into $B(\lambda)$ blocks $\mathcal{B}_1, \ldots, \mathcal{B}_{B(\lambda)}$, $B(\lambda)$ depending on λ, indexed such that

$$(w_j - \sum_{J_k \in \mathcal{S}_j} \lambda_{jk} + \sum_{J_k \in \mathcal{P}_j} \lambda_{kj})/p_j = c_b, \quad \text{for each } J_j \in \mathcal{B}_b,$$

with $c_1 > \cdots > c_{B(\lambda)} > 0$.

THEOREM 2.3. *Any vector λ satisfying the termination conditions induced a decomposition of $\mathcal{J}$ into $B(\lambda)$ blocks $\mathcal{B}_1, \ldots, \mathcal{B}_{B(\lambda)}$, such that, if $(J_j, J_k) \in A$ and $J_k \in \mathcal{B}_b$, then*

$$J_j \in \mathfrak{B}_1 \cup \cdots \cup \mathfrak{B}_b,$$
$$\lambda_{jk} = 0 \quad \textit{if } J_j \in \mathfrak{B}_1 \cup \cdots \cup \mathfrak{B}_{b-1}.$$

PROOF. If one of these claims is not true, then we can still identify an ascent direction, contradicting the assumption that the termination conditions are satisfied. □

Such a decomposition, i.e., a decomposition induced by a vector satisfying the termination conditions, is called a *dual decomposition*. Both λ^* and $\bar{\lambda}$ induce dual decompositions. For our example, the dual decomposition induced by $\bar{\lambda}$ consists of three blocks: $\mathfrak{B}_1 = \{J_3\}$, $\mathfrak{B}_2 = \{J_2, J_4\}$, and $\mathfrak{B}_3 = \{J_1, J_5, J_6, J_7, J_8, J_9, J_{10}\}$, with $c_1 = 9$, $c_2 = 11/12$, and $c_3 = 6/7$, respectively.

2.2. APPROXIMATION

We present an approximation algorithm that exploits the agreeable structure of the dual decomposition induced by $\bar{\lambda}$. For $b = 1, \ldots, B(\bar{\lambda})$, let σ_b be a feasible sequence for the jobs in $\mathfrak{B}_b$. From Theorem 2.3, we derive the following.

COROLLARY 2.1. *The sequence* $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{B(\bar{\lambda})})$ *is a feasible sequence for the overall problem.* □

If each σ_b is optimal for the $1|prec|\Sigma_{J_j \in \mathfrak{B}_b} w_j C_j$ problem ($b = 1, \ldots, B(\bar{\lambda})$), then we have the best such σ. From a theoretical point of view, each $1|prec|\Sigma_{J_j \in \mathfrak{B}_b} w_j C_j$ problem is as hard as the overall problem; from a practical point of view, each problem, being of smaller dimension, is simpler. Dynamic programming, when using a compact labeling scheme as proposed by Schrage and Baker [1978] and Lawler [1979], solves small instances quickly. If the size of a block is too large for the application of dynamic programming, then we resort to the tree-optimal-heuristic, presented by Morton and Dharan [1978], to find an approximate solution. However, even if the dual decomposition is induced by λ^* and σ is composed of optimal subsequences, then we still have no guarantee that σ is an optimal sequence; all optimal sequences may have been excluded by the dual decomposition.

For the example, the optimal sequences for the first two blocks are trivial: $\sigma_1 = (J_3)$, and $\sigma_2 = (J_2, J_4)$; using dynamic programming, we find $\sigma_3 = (J_1, J_7, J_5, J_9, J_6, J_8, J_{10})$; the tree-optimal-heuristic gives the same sequence. We obtain $\sigma = (J_3, J_2, J_4, J_1, J_7, J_5, J_9, J_6, J_8, J_{10})$, having total cost 1530.

We tested the approximation algorithm on problems with 20, 30, ..., 100 jobs. The processing times were drawn from the uniform distribution [1,100]; the weights were generated from the uniform distribution [1,10]. The precedence graph was induced by the probability P with which each arc (J_j, J_k) with $j < k$ was included. The graph obtained in this way was then subsequently stripped down to its transitive reduction. We generated problems for $P = 0.001$, 0.02,

0.04, 0.06, 0.08, 0.10, 0.15, 0.20, 0.30, and 0.50. For each combination of n and P we generated five problems; hence, 45 problems were generated for each value P. This procedure parallels Potts' procedure to generate instances. Furthermore, we solved each subproblem to optimality if less than 15000 labels were needed; otherwise, we used the tree-optimal-heuristic.

P	Tree-Opt-Heuristic	Dual Decomposition
0.001	0.007 (42)	0.007 (42)
0.02	0.074 (15)	0.069 (15)
0.04	0.516 (8)	0.248 (10)
0.06	1.214 (2)	0.675 (2)
0.08	1.446 (1)	1.076 (1)
0.10	2.040 (2)	1.518 (4)
0.15	2.252 (0)	2.024 (0)
0.20	2.551 (0)	2.113 (2)
0.30	4.111 (0)	3.733 (2)
0.50	4.334 (2)	4.116 (3)

TABLE 2.2. Experimental results. For each value of P, the average relative deviation of the upper bound from the lower bound is given. The figures within brackets indicate the number of times out of 45 that the upper bound equalled the lower bound.

In Table 2.2, the computational results are given. Potts already pointed out that the relative difficulty of an instance depends more on $|A|$ than on n. We have therefore classified the results according to the value P rather than n. For each P, we present the relative deviation between upper bound and lower bound for both the tree-optimal-heuristic and the dual-decomposition approach. Within brackets we indicate for how many problems (out of 45) the upper bound equalled the lower bound; this figure gives the number of times we found a provably optimal solution. On the average, the dual-decomposition algorithm outperforms the tree-optimal-heuristic approach for any problem class. For the 450 instances altogether, the tree-optimal-heuristic produced only 16 solutions that were better; moreover, each of these was only marginally better.

The tree-optimal-heuristic requires $O(n\,|A|)$ time, and is therefore sensitive to instances with many precedence constraints. The running time of the dual-decomposition approximation algorithm mainly depends on the number of calls on the dynamic programming procedure and the maximum label number. We have coded both algorithms in the computer language C; all experiments were conducted on a Compaq-386/20 Personal Computer. For $n \leqslant 40$, the tree-optimal-heuristic needed a few seconds at most. On the average, our approximation required only slightly more computation; there were, however, occasional peaks due to high labels in the dynamic programming subroutine. For $n \geqslant 60$, the tree-optimal-heuristic needs about twice or three times as much computation

time as the dual-decomposition algorithm; even the peaks of the latter remain below the average of the former.

Potts also points out that small and large values of P generate relatively easy problems. For small P, only few precedence constraints are involved; for large P, most disjunctive constraints are settled. Our results support the claim for small P: the duality gap is very small. Since the optimal-tree-heuristic generates good approximate solutions for all values of P [Potts, 1985A], there are two possible explanations for the growth of the gap between upper bound and upper bound for larger values of P. It may be that the ascent direction method produces worse approximate solutions in case P is large; it is more likely, however, that the duality gap is an increasing function of P.

2.3. Primal decomposition

For $b = 1, \ldots, B$, let $\sigma_b{}^*$ denote an optimal sequence for the problem $1|prec|\Sigma_{J_j \in \mathcal{J}_b} w_j C_j$, where $\mathcal{J}_b \subseteq \mathcal{J}$. A decomposition of the job set $\mathcal{J}$ into B mutually disjoint subsets $\mathcal{J}_1, \ldots, \mathcal{J}_B$ is said to be a *primal decomposition* if the sequence $\sigma = \sigma_1{}^*, \ldots, \sigma_B{}^*$ is optimal for the overall $1|prec|\Sigma w_j C_j$ problem. We already mentioned that a dual decomposition may exclude all optimal sequences; a dual decomposition only suggests a primal decomposition. In this section, we try to establish to what extent a dual decomposition coincides with a primal decomposition.

If a dual decomposition excludes all optimal solutions, then there are at least two jobs belonging to different blocks with no path in A between them for which the processing order should be reversed. Suppose $J_j \in \mathcal{B}_b$ and $J_k \in \mathcal{B}_{b+m}$ $(m > 0)$ are such jobs. In all feasible sequences obtained by the dual-decomposition approach, J_j precedes J_k; but in all optimal sequences, J_k precedes J_j. Hence, the arc (J_k, J_j) can be added to the arc set A without impunity. Let problem $(L_\lambda(k,j))$ be the Lagrangian problem for the arc set $A \cup (J_k,J_j)$, and let $\lambda(k,j) \geq 0$ be a vector of Lagrangian multipliers. Since the arc (J_k,J_j) does not exclude the optimal solution, $L(\lambda(k,j))$ is still a lower bound on the optimal solution, for any $\lambda(k,j) \geq 0$.

This observation gives rise to the following theorem. Let $\mathcal{B}_1, \ldots, \mathcal{B}_B$ be the blocks of some dual decomposition.

THEOREM 2.4. *If there are two jobs $J_j \in \mathcal{B}_b$ and $J_k \in \mathcal{B}_{b+m}$ $(b = 1, \ldots, B-1$, $m = 1, \ldots, B-b)$ with no path in A between them for which*

$$L(\lambda(k,j)) > UB - 1,$$

then J_j precedes J_k in all optimal solutions for the $1|prec|\Sigma w_j C_j$ problem. □

If Theorem 2.4 applies to all pairs of such jobs, then the dual decomposition is a primal decomposition; in fact, due to transitivity, it only has to apply to specific pairs of jobs.

COROLLARY 2.1. *If for each pair of jobs $J_j \in \mathcal{B}_b$ and $J_k \in \mathcal{B}_{b+m}$ $(b = 1, \ldots, B-1$, $m = 1, \ldots, B-b)$ such that*

(*i*) *there is no path in A from J_j to J_k,*

(*ii*) *J_j has no successors in $\mathfrak{B}_b \cup \cdots \cup \mathfrak{B}_{b+m-1}$, and*

(*iii*) *J_k has no predecessors in $\mathfrak{B}_{b+1} \cup \cdots \cup \mathfrak{B}_{b+m}$,*

we have that $L(\lambda(k,j)) > UB - 1$ for some $\lambda(k,j) \geqslant 0$, then the dual decomposition is a primal decomposition. □

Accordingly, if the dual decomposition induces a primal decomposition and if UB is associated with the sequence that is composed of optimal subsequences, then UB is the optimal solution value of the $1 | prec | \Sigma w_j C_j$ problem.

COROLLARY 2.2. *If for some block $\mathfrak{B}_b$, each pair of jobs $J_j \in \mathfrak{B}_b$ and $J_k \in \mathfrak{B}_{b+m}$ ($m = 1, \ldots, B-b$) such that*

(*i*) *there is no path in A from J_j to J_k,*

(*ii*) *J_j has no successors in $\mathfrak{B}_b \cup \cdots \cup \mathfrak{B}_{b+m-1}$, and*

(*iii*) *J_k has no predecessors in $\mathfrak{B}_{b+1} \cup \cdots \cup \mathfrak{B}_{b+m}$,*

satisfies $L(\lambda(k,j)) > UB - 1$ for some $\lambda(k,j) \geqslant 0$, then the subsets $\mathfrak{B}_1 \cup \cdots \cup \mathfrak{B}_b$ and $\mathfrak{B}_{b+1} \cup \cdots \cup \mathfrak{B}_B$ constitute a primal decomposition of $\mathcal{J}$. □

In this case, we say that the dual decomposition concurs *partly* with a primal decomposition.

Whether we succeed to establish that a dual decomposition concurs partly or completely with a primal decomposition depends on the quality of the lower bounds $L(\lambda(k,j))$. From this point of view, we like to have available the vector of optimal Lagrangian multipliers for problem ($L_\lambda(k,j)$); let $\lambda^*(k,j)$ denote this vector. Of course, $\lambda^*(k,j)$ is as difficult to find as the vector λ^*. However, an ascent direction method to approximate $\lambda^*(k,j)$ is readily available: we apply the direction method for problem (D), adjusted for the additional arc (J_k,J_j), using as initial vector $\lambda(k,j)^{(0)}$ obtained as $\lambda(k,j)^{(0)}_{ih} = \bar{\lambda}_{ih}$ for each $(J_i,J_h) \in A$ and $\lambda(k,j)^{(0)}_{kj} = 0$. We note that $L(\lambda(k,j)^{(0)}) = L(\bar{\lambda})$. At $\lambda(k,j)^{(0)}$, all primitive directional derivatives are non-positive but one: we have $l^+_{kj}(\lambda(k,j)^{(0)}) > 0$; increasing $\lambda(k,j)_{kj}$ is an ascent direction. If J_j and J_k belong to blocks that lie far apart from each other, then the Lagrangian lower bound corresponding with the point where the sign of this directional derivative changes may already exceed $UB - 1$. This Lagrangian lower bound is conveniently computed; this is stipulated in the next theorem, where $p(\mathfrak{B}_b)$ is defined as $p(\mathfrak{B}_b) = \Sigma_{J_i \in \mathfrak{B}_b} p_i$.

THEOREM 2.5. *If there are two jobs $J_j \in \mathfrak{B}_b$ and $J_k \in \mathfrak{B}_{b+m}$ ($b = 1, \ldots, B(\bar{\lambda})-1$, $m = 1, \ldots, B(\bar{\lambda})-b$) for which there is no path in A from J_j to J_k such that*

$$L(\bar{\lambda}) + (c_b - c_{b+m}) p_j p_k + \sum_{i=b+1}^{l} (c_b - c_i) p(\mathfrak{B}_i) p_j + \sum_{i=l+1}^{b+m-1} (c_i - c_{b+m}) p(\mathfrak{B}_i) p_k$$

exceeds $UB - 1$, where l is the largest index with $c_l \geqslant (p_j c_b + p_k c_{b+m})/(p_k + p_j)$, then J_j precedes J_k in all optimal solutions for the $1 | prec | \Sigma w_j C_j$ problem.

PROOF. The validation of this theorem requires the same logic applied in the proof of Theorem 2.2. □

If Theorem 2.5 does not apply, then we run the ascent direction algorithm until no ascent directions can be found anymore; upon termination, we get the vector $\overline{\lambda}(k,j)$.

We now work out the effects of these theorems on our example. According to Corollary 2.2, we need consider only the pairs (J_3,J_2) and (J_3,J_1) in order to decompose the jobs into $\mathfrak{B}_1$ on the one hand and $\mathfrak{B}_2 \cup \mathfrak{B}_3$ on the other. Including (J_2,J_3) in A, we get, by use of Theorem 2.5, that $L(\lambda^*)+(c_1-c_2)p_2p_3=1526.69+(9-11/12)\cdot 1\cdot 9>1529$; we conclude that J_3 precedes J_2 in any schedule with cost less than 1530. Similarly, if we include (J_1,J_3) in A, then, according to Theorem 2.5 (where we have $l=1$), we must verify if

$$L(\lambda^*) + (c_1-c_3)p_1p_3 + (c_2-c_3)p(\mathfrak{B}_2)p_1 > UB-1.$$

This is so; hence, J_3 must precede J_1, implying that we may decompose the job set into subsets $\mathfrak{B}_1$ and $\mathfrak{B}_2 \cup \mathfrak{B}_3$. We must consider the pairs (J_4,J_1), (J_4,J_5), and (J_4,J_8) to separate the blocks $\mathfrak{B}_2$ and $\mathfrak{B}_3$. More than one iteration in the ascent direction procedure is required. Since $L(\overline{\lambda}(1,4))$, $L(\overline{\lambda}(5,4))$, and $L(\overline{\lambda}(8,4))$ exceed 1529, we conclude that the dual decomposition concurs with a primal decomposition. Furthermore, the schedule with value 1530 is optimal, since it was obtained from the optimal sequences for the individual blocks.

The theorems and corollaries presented in this section are applicable in a preprocessing phase in conjunction with any existing branch-and-bound algorithm. Their main purpose is to derive additional precedence constraints and to primally decompose the problem in order to reduce the size of the search tree.

2.4. THE TOTAL WEIGHTED TARDINESS PROBLEM

We claimed earlier that the methodology applied to analyze the $1|prec|\Sigma w_jC_j$ problem is a generic tool for the analysis of single-machine problems that can be cast in the format presented at the beginning of this chapter. The effectiveness of the dual decomposition method, however, largely depends on the structure of the problem and the nature of the dualized constraints. We consider here in brief the total weighted tardiness problem; at first sight, it may not be so apparent that the generic format also covers this problem.

The problem setting for the total weighted tardiness problem is the same as for the $1|prec|\Sigma w_jC_j$ problem, albeit that the jobs are now independent and have a due date d_j by which they should be completed. Given a schedule, the tardiness T_j of J_j is defined as $T_j=\max\{C_j-d_j,0\}$. The objective is to find a schedule that minimizes total weighted tardiness, that is, $\Sigma_{j=1}^n w_jT_j$.

The $1||\Sigma w_jT_j$ problem is $\mathfrak{NP}$-hard in the strong sense. The best branch-and-bound algorithm is due to Potts and Van Wassenhove [1985], and solves instances up to 50 jobs. Potts and Van Wassenhove [1988] also investigate the performances of a number of approximation algorithms for this problem.

The problem can be formulated as follows. Determine job tardinesses $T_1, \ldots, T_n$ and job completion times $C_1, \ldots, C_n$ that minimize

$$\sum_{j=1}^{n} w_j T_j$$

subject to

$$T_j \geqslant C_j - d_j, \quad \text{for } j = 1, \ldots, n, \tag{2.6}$$

$$T_j \geqslant 0, \quad \text{for } j = 1, \ldots, n, \tag{2.7}$$

$$\text{the capacity and availability of the machine.} \tag{2.8}$$

The conditions (2.6) and (2.7) simply reflect the definition of job tardiness. It is easy to verify that this formulation, first employed by Potts and Van Wassenhove [1985], matches the generic formulation.

Using a given vector $\lambda = (\lambda_1, \ldots, \lambda_n) \geqslant 0$ of Lagrangian multipliers to dualize the conditions (2.6), we obtain the following Lagrangian problem: determine the value $L(\lambda)$, which is the minimum of

$$\sum_{j=1}^{n} (w_j - \lambda_j) T_j + \sum_{j=1}^{n} \lambda_j C_j - \sum_{j=1}^{n} \lambda_j d_j$$

subject to the conditions (2.7) and (2.8).

The conditions (2.7) only affect the first component of the Lagrangian objective function; the conditions (2.8) affect only the second component. Accordingly, the Lagrangian problem decomposes into two subproblems. Requiring that $w_j - \lambda_j \geqslant 0$ for each j $(j = 1, \ldots, n)$, we minimize the first component by setting $T_j = 0$ for each j. After all, we get $T_j = \infty$ if $w_j - \lambda_j < 0$, resulting in $L(\lambda) = -\infty$. The second component reduces to the $1 \,||\, \Sigma \lambda_j C_j$ problem; it is simply solved by scheduling the jobs in order of non-increasing values λ_j / p_j in the interval $[0, \Sigma_{j=1}^{n} p_j]$.

Potts and Van Wassenhove do not solve the Lagrangian dual problem to optimality (which is possible in polynomial time by use of the ellipsoid method), but apply a specific one-shot method to set the Lagrangian multipliers (cf. Section 1.3.3.4). On the other hand, a quick ascent direction method, similar to the one for the $1 \,|\, prec \,|\, \Sigma w_j C_j$ problem, is easily developed. The termination conditions for an ascent direction method provide that the dual decomposition induced by $\bar{\lambda}$ has the following structure. The job set $\mathcal{J}$ is decomposed into blocks $\mathcal{B}_1, \ldots, \mathcal{B}_{B(\bar{\lambda})}$ such that for each $J_j \in \mathcal{B}_b$ we have

$$d_j \leqslant \sum_{J_k \in \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_b} p_k,$$

and

$$\bar{\lambda}_j = w_j \text{ if } d_j < p_j + \sum_{J_k \in \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_{b-1}} p_k.$$

Furthermore, we must have

$$\bar{\lambda}_j = \lambda_j^* = 0 \text{ if } d_j \geqslant \sum_{k=1}^{n} p_k.$$

This block structure can be exploited for the design of approximative and enumerative methods for the $1||\Sigma w_j T_j$ problem in the same fashion as for the $1|prec|\Sigma w_j C_j$ problem. For instance, the combination of optimal (or feasible) sequences for the individual blocks yields a feasible schedule for the entire problem. It remains to be seen, however, whether such algorithms outperform existing algorithms.

3

Flow-Shop Scheduling

We consider the $F2||\Sigma C_j$ problem, and develop a branch-and-bound algorithm for its solution that employs a Lagrangian lower bound. In Section 3.1, we introduce the problem. In Section 3.2, we formulate it in terms of the job completion times on both machines. The initial Lagrangian relaxation decomposes the problem into two single-machine scheduling problems. If we add a constraint that is redundant for the primal problem, then the Lagrangian problem becomes a linear ordering problem. Although the linear ordering problem is, in general, $\mathcal{NP}$-hard, it is polynomially solvable in case of appropriate choices for the Lagrangian multipliers. The best choice within this class yields a lower bound that dominates previous bounds. In fact, the bounds that have been proposed before correspond to particular choices of the multipliers. It is shown how the new lower bound can be strengthened and how precedence constraints can be derived from the Lagrangian problem. Section 3.3 presents dominance criteria to restrict the search tree. A complete description of the branch-and-bound algorithm and a presentation of some computational results are given in Section 3.4. These results show that the proposed algorithm outperforms the previously best method. Finally, we demonstrate in Section 3.5 how a similar bound is obtained for the general case of m machines.

3.1. Introduction

An m-machine flow shop is described as follows. There is a set of m machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ that are continuously available from time 0 onwards for processing a set of n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each machine can handle no more than one job at a time. Each job consists of a chain of m operations. The ith operation of job J_j has to be processed on machine M_i during a positive uninterrupted time p_{ij} ($i = 1, \ldots, m$, $j = 1, \ldots, n$). Each job can be executed by at most one machine at a time, implying that operations of the same job may not

overlap in their execution. Note that the jobs go through the machines in the same order. A *schedule* specifies a completion time C_{ij} for the ith operation ($i = 1, \ldots, m$) of each J_j ($j = 1, \ldots, n$) such that the above conditions are met. The completion time of job J_j is then simply C_{mj}.

A voluminous part of flow-shop research has been focused on the minimization of the length of a schedule or the so-called makespan. Gupta and Dudek [1971], however, plead that criteria reflecting the cost of each job individually have a better economic interpretation than the makespan objective has. We consider here the $F2 \| \Sigma C_j$ problem, that is, the problem of minimizing the sum of the job completion times in the 2-machine flow shop.

It is well known that for this problem it suffices to optimize over all permutation schedules [Conway, Maxwell, and Miller, 1967]. A *permutation schedule* is a schedule in which every machine has the same job sequence. Ignall and Schrage [1965], the first to study this problem, present a branch-and-bound algorithm that is based on two lower bounds. Their paper is a classic, as they are the first to describe a branch-and-bound algorithm for a machine scheduling problem. The heuristics presented by Krone and Steiglitz [1974] are applied by Kohler and Steiglitz [1975] in further developing and testing the Ignall and Schrage algorithm. Garey, Johnson, and Sethi [1976] prove that the problem is $\mathcal{NP}$-hard in the strong sense by a reduction from 3-PARTITION.

For the general case of m machines, Szwarc [1983] derives some properties of an optimal schedule, and identifies a class of well-solvable cases. A more elaborate treatment of well-solvable cases is found in Adiri and Amit [1984]. Bansal [1977] extends the Ignall and Schrage lower bounds to the m-machine case.

3.2. FORMULATION AND RELAXATION

We give a formulation of the $F2 \| \Sigma C_j$ problem in terms of the completion times of the operations. The problem, in the remainder of this chapter referred to as problem (P), is then as follows: determine completion times C_{ij} ($i = 1, 2$, $j = 1, \ldots, n$) that minimize

$$\sum_{j=1}^{n} C_{2j} \tag{P}$$

subject to

the precedence constraints between the operations of the jobs, (3.1)

the capacity constraints of the machines, (3.2)

the availability constraints of the machines. (3.3)

The conditions (3.1) are formulated as

$$C_{2j} \geqslant C_{1j} + p_{2j}, \qquad \text{for } j = 1, \ldots, n.$$

We introduce a vector of multipliers $\lambda = (\lambda_1, \ldots, \lambda_n) \geqslant 0$ to dualize the conditions (3.1). Lagrangian relaxation of these conditions yields the following Lagrangian problem, referred to as problem (L_λ): for a given $\lambda \geqslant 0$, determine the value $L(\lambda)$ which is the minimum of

$$\sum_{j=1}^{n}(\lambda_j C_{1j} + (1-\lambda_j)C_{2j} + \lambda_j p_{2j})$$

subject to (3.2) and (3.3).

From Section 1.3, we know the value $L(\lambda)$ provides a lower bound to (P) for any $\lambda \geq 0$. In order to prevent $L(\lambda)$ from becoming arbitrarily small, we must require that $\lambda_j \leq 1$ for $j = 1, \ldots, n$. After all, if $\lambda_j > 1$ for some j ($j = 1, \ldots, n$), then we get $C_{2j} = \infty$, thereby disqualifying the strength of the lower bound.

The precedence relations between the operations are absent in the Lagrangian problem; operations belonging to the same job can now be processed simultaneously. The Lagrangian problem decomposes into two single-machine problems, which are solved by Smith's [1956] ratio rule: schedule the jobs on M_1 and M_2 in order of non-increasing ratios λ_j / p_{1j} and $(1-\lambda_j)/p_{2j}$, respectively. The Lagrangian dual problem of finding the vector of Lagrangian multipliers that gives the best lower bound can be solved to optimality by use of the ellipsoid method (see Section 1.3.3.1). In addition, an ascent direction method for approximating the optimal solution of the Lagrangian dual problem is easily developed along the lines of Chapter 2.

We choose another approach. Its gist lies in imposing the restriction to solve (L_λ) over all permutation schedules. This condition is redundant for the primal problem, but is not redundant for the Lagrangian problem; hence, it may increase the value $L(\lambda)$. We will choose the multiplier vector λ in such a way that solving (L_λ) over all permutation schedules can still be accomplished in polynomial time.

To that end, we first reformulate the problem of solving (L_λ) over all permutation schedules as a *linear ordering problem*. The linear ordering problem is the following: given an $n \times n$ matrix $A = (a_{jk})$ of weights, find a permutation σ of $\{1, \ldots, n\}$ that maximizes the sum

$$\sum_{(j,k):\sigma(j)<\sigma(k)} a_{jk},$$

where $\sigma(j)$ denotes the position of element j in the sequence σ. In our application, we identify $\sigma(j)$ with the position of job J_j. Since we have in problem (L_λ) that

$$C_{ij} = \sum_{k:\sigma(k)\leq\sigma(j)} p_{ik}, \tag{3.4}$$

it follows that

$$\sum_{j=1}^{n}(\lambda_j C_{1j} + (1-\lambda_j)C_{2j})$$

$$= \sum_{j=1}^{n}\left[\lambda_j \sum_{k:\sigma(k)\leq\sigma(j)} p_{1k}\right] + \sum_{j=1}^{n}\left[(1-\lambda_j) \sum_{k:\sigma(k)\leq\sigma(j)} p_{2k}\right]$$

$$= \sum_{j=1}^{n}\sum_{k=1}^{n}\left[\lambda_j p_{1k} (1-\lambda_j) p_{2k}\right] - \sum_{j=1}^{n} \sum_{k:\sigma(j)<\sigma(k)} \left[\lambda_j p_{1k} + (1-\lambda_j) p_{2k}\right].$$

Hence, solving (L_λ) over all permutation schedules is equivalent to finding a

permutation σ that maximizes

$$\sum_{(j,k):\sigma(j)<\sigma(k)} (\lambda_j p_{1k} + (1-\lambda_j)p_{2k}).$$

Bergmans [1972] and Pratt [1972] show, using an adjacent pairwise interchange argument, that the linear ordering problem is polynomially solvable for two special cases; see also Picard and Queyranne [1982] and Kolen [1986]. If the weights are in product form, i.e., $a_{jk} = x_j y_k$, the linear ordering problem is solved by ordering the elements according to non-increasing ratios x_j/y_j. This order is exactly induced by Smith's rule for the $1||\Sigma w_j C_j$ problem. The linear ordering problem is also solved in polynomial time if the weights are in sum form, i.e., $a_{jk} = x_j + y_k$; an optimal permutation is then obtained by ordering the elements according to non-increasing values $x_j - y_j$. The choice $\lambda_j = c$ for each j $(j = 1, \ldots, n)$, for some constant $c\,(0 \leq c \leq 1)$, converts (3.4) into an even simpler polynomially solvable case of the linear ordering problem: we get the form $a_{jk} = y_k$, solved by ordering the elements according to non-decreasing values y_k. Hence, for those particular values of λ, solving problem (L_λ) over all permutation schedules amounts to scheduling the jobs in order of non-decreasing values $cp_{1k} + (1-c)p_{2k}$. The values $c = 0$ and $c = 1$ render exactly the Ignall and Schrage lower bounds; in fact, these bounds result from applying Smith's rule to each of the machines separately.

In the remainder of this chapter, the notation (L_c) refers to the problem (L_λ) with $\lambda_j = c$ for each j $(j = 1, \ldots, n)$, and $L(c)$ denotes its optimal objective value.

3.2.1. *Solving the Lagrangian dual*

We are interested in solving the restricted Lagrangian dual, referred to as problem (D), that is, in finding the value c $(0 \leq c \leq 1)$ that maximizes $L(c)$:

$$\max\{\min \sum_{j=1}^{n} [C_{2j} + c(C_{1j} + p_{2j} - C_{2j})] \mid 0 \leq c \leq 1\}. \tag{D}$$

The function $L : c \to L(c)$ has nice properties that make it easy to solve problem (D) to optimality.

First, there is only one variable involved. Second, L is a continuous, concave, and piecewise linear function in c (see Section 1.3.3). Hence, an optimal solution is found in one of the breakpoints of the function. These breakpoints are characterized in the following way. Job J_j is said to be *c-preferable* to J_k if $cp_{1j} + (1-c)p_{2j} < cp_{1k} + (1-c)p_{2k}$; this means that J_j is scheduled before J_k in any optimal solution to problem (L_c). If J_j is c-preferable to J_k for all c $(0 \leq c \leq 1)$, then J_j is *strongly preferable* to J_k. For each pair (J_j, J_k) without a strong preference relation, we define the *critical value* as the value c for which both jobs are equally preferable, i.e., $cp_{1j} + (1-c)p_{2j} = cp_{1k} + (1-c)p_{2k}$. These critical values are precisely the breakpoints of the function L. Third, the problem (L_c) is solvable in polynomial time for given c. Hence, an ascent direction algorithm is easy to develop; furthermore, such an algorithm gives an optimal solution, as only one variable is involved.

The procedure to solve (D) is the following. Find the $O(n^2)$ critical values and sort them in non-decreasing order. Starting with an arbitrary critical value $\bar{c}$, we solve ($L_{\bar{c}}$), settling ties arbitrarily. Subsequently, we evaluate the directional derivatives at $\bar{c}$. Inspecting the Lagrangian dual problem, we derive that $l^+(\bar{c})$, the directional derivative for increasing $\bar{c}$, is

$$l^+(\bar{c}) = \sum_{j=1}^{n} (C_{1j} + p_{2j} - C_{2j}),$$

and that $l^-(\bar{c})$, the directional derivative for decreasing $\bar{c}$, is

$$l^-(\bar{c}) = \sum_{j=1}^{n} (C_{2j} - p_{2j} - C_{1j}).$$

If both $l^+(\bar{c}) \leqslant 0$ and $l^-(\bar{c}) \leqslant 0$, then there is no direction of ascent: $\bar{c}$ is optimal. Otherwise, we perform binary search over the appropriate interval. The appropriate interval is $[\bar{c}, 1]$ in case $l^+(\bar{c}) > 0$, i.e., in case increasing $\bar{c}$ is an ascent direction; the appropriate interval is $[0, \bar{c}]$ in case $l^-(\bar{c}) > 0$, i.e., in case decreasing $\bar{c}$ is an ascent direction. Since sorting the critical values takes $O(n^2 \log n)$ time, binary search over $O(n^2)$ points takes $O(\log n)$ iterations, and each problem (L_c) requires $O(n \log n)$ time, the Lagrangian dual problem is solved in $O(n^2 \log n)$ time. Since $\max_{0 \leqslant c \leqslant 1} L(c) \geqslant \max\{L(0), L(1)\}$, problem (D) produces a lower bound that dominates the Ignall and Schrage lower bounds.

3.2.2. *Strengthening the lower bound*

Let c^* be the value of c that solves problem (D). Suppose now that the multiplier vector λ is perturbed in the jth component by a term Δ_j, i.e., $\lambda_j = c^* + \Delta_j$; suppose further that this perturbation does not change the processing order. Obviously, the lower bound would be affected by the term

$$\Delta_j (C_{1j} + p_{2j} - C_{2j}) \tag{3.5}$$

if the value $c^* p_{1j} + (1 - c^*) p_{2j}$ is not tied, i.e., if the position of J_j is the same in all optimal solutions to problem (L_{c^*}).

Let $a_{jk} = \lambda_j p_{1k} + (1 - \lambda_j) p_{2k}$. If λ_j were perturbed by Δ_j, then the jth row in the weight matrix A for the linear ordering problem would become $a_{jk} + \Delta_j(p_{1k} - p_{2k})$, for $k = 1, \ldots, n$. The issue now is to determine the range for Δ_j such that the optimal solution to the perturbed problem is the same as to (L_{c^*}). Recall that the choice $\lambda_j = c$ for each j ($j = 1, \ldots, n$) implies for the solution of problem (L_c) that

$$a_{kj} > a_{jk} \Leftrightarrow \sigma(j) > \sigma(k), \qquad \text{for } k = 1, \ldots, n.$$

Hence, a sufficient condition to ensure that the optimal solution remains the same is that for each k ($k = 1, \ldots, n$, $k \neq j$) we have

$$a_{kj} \geqslant a_{jk} + \Delta_j(p_{1k} - p_{2k}) \quad \text{if } \sigma(j) > \sigma(k), \tag{3.6}$$

$$a_{kj} \leqslant a_{jk} + \Delta_j(p_{1k} - p_{2k}) \quad \text{if } \sigma(j) < \sigma(k). \tag{3.7}$$

The next step is therefore to compute for each k, $k \neq j$, the value δ_{jk} for which the values a_{kj} and $a_{jk} + \delta_{jk}(p_{1k} - p_{2k})$ coincide, if such a value exists. From this we get

$$\delta_{jk} = (a_{kj} - a_{jk})/(p_{1k} - p_{2k}) \quad \text{if } p_{1k} \neq p_{2k}.$$

Defining $\Delta_j^+ = \min_k\{\delta_{jk} \mid \delta_{jk} \geqslant 0$ and $p_{1k} \neq p_{2k}\}$ and $\Delta_j^- = \max_k\{\delta_{jk} \mid \delta_{jk} \leqslant 0$ and $p_{1k} \neq p_{2k}\}$, respectively, we conclude that as long as λ_j is perturbed by Δ_j with $\Delta_j^- \leqslant \Delta_j \leqslant \Delta_j^+$, the optimal solution to (L_{c^*}) is also optimal to the perturbed problem. Therefore, the current lower bound can be improved by maximizing (3.5) subject to $\Delta_j^- \leqslant \Delta_j \leqslant \Delta_j^+$ and $0 \leqslant \lambda_j + \Delta_j \leqslant 1$. Hence, the Lagrangian weights are perturbed in the following way:

$$(a) \ \lambda_j \leftarrow \min\{c^* + \Delta_j^+, 1\} \quad \text{if } C_{1j} + p_{2j} > C_{2j},$$

$$(b) \ \lambda_j \leftarrow \max\{c^* + \Delta_j^-, 0\} \quad \text{if } C_{1j} + p_{2j} < C_{2j}.$$

This analysis can consecutively be performed for each J_j with untied value $c^* p_{1j} + (1-c^*)p_{2j}$. It takes $O(n^2)$ time altogether. We note that the final lower bound depends on the order in which the multipliers have been adjusted.

3.2.3. Precedence constraints

Job J_j is said to have *precedence* to job J_k, denoted by $J_j \to J_k$, if there is an optimal solution in which J_j precedes J_k. This type of precedence constraint is not given *a priori*, but it is derived *a posteriori* to reduce the set of relevant schedules. We try to derive such precedence constraints from the Lagrangian problem. The technique for this is based upon the following concept. Let ($L_c(j,k)$) denote the problem (L_c) to which we added the constraint that J_j precedes J_k, while J_k is c-preferable to J_j; let $L_{jk}(c)$ denote its optimal objective value. Clearly, we have $L_{jk}(c) > L(c)$. If $L_{jk}(c)$ exceeds a known upper bound, then there is obviously an optimal solution to (P) in which $J_k \to J_j$. We only have to deal with the question whether ($L_c(j,k)$) is polynomially solvable. Fortunately, this is the case. A single-machine result from Monma and Sidney [1979] for objective functions that possess the adjacent pairwise interchange property applies to problem ($L_c(j,k)$). This result, proved by an interchange argument, clears the way for solving ($L_c(j,k)$) in a straightforward way.

THEOREM 3.1. *For the problem ($L_c(j,k)$) with J_k c-preferable to J_j, there is an optimal schedule in which J_k is sequenced immediately after J_j.* □

An optimal sequence for ($L_c(j,k)$) can then be obtained in the following way. Start by scheduling all jobs as in the solution of problem (L_c) and remove J_j and J_k from this sequence. We call this sequence π. The *module* $\{J_j, J_k\}$ is then inserted just before the first job J_l in π for which $2(cp_{1l} + (1-c)p_{2l}) > c(p_{1j} + p_{1k}) + (1-c)(p_{2j} + p_{2k})$. If no such job exists, then $\{J_j, J_k\}$ is scheduled last. This condition stems from comparing the objective values for the partial sequences $J_jJ_kJ_l$ and $J_lJ_jJ_k$. The lower bound $L_{jk}(c)$ can be strengthened in the same spirit as described in Section 3.2.2.

3.3. Dominance criteria

A node at level l of the branch-and-bound procedure corresponds to an initial partial sequence π in which l jobs have been put in the first l positions. For each node at level l, at most $n-l$ descendant nodes are created, one for every job without unscheduled predecessors. Let $C_i(\pi)$ be the completion time of the last job in the sequence π on M_i. The sum of the job completion times on M_2 of the jobs in π is denoted by $TC(\pi)$. Then there is no need to branch from a node having π as an initial sequence if there is a permutation π^* of the jobs in π, $\pi^* \neq \pi$, that satisfies the following conditions:

$$TC(\pi^*) \leqslant TC(\pi), \tag{3.8}$$

$$C_2(\pi^*) \leqslant \max\{\, C_2(\pi), C_1(\pi) + \min_{J_j \notin \pi} p_{1j} \,\}. \tag{3.9}$$

In this case, we say that the sequence π is *dominated* by π^*. The condition (3.9) ensures that the unscheduled jobs can start on M_2 at least as soon in case of π^* as initial sequence as in case of π. Of course, finding out whether a given permutation π is dominated or not is as hard as the original problem. A *dominance rule* gives an easy-to-check sufficient condition for the existence of dominance.

The following three rules are checked as soon as we are about to add a new job J_j to the current initial sequence. The *dynamic programming dominance* criterion is probably the most obvious one: a node that corresponds with the sequence $\pi = \rho J_k J_j$ can be eliminated if the sequence π is dominated by the sequence $\pi^* = \rho J_j J_k$; ρ is here a subsequence of jobs.

The second one reschedules the jobs in $\pi = \rho J_j$ into π^* according to Johnson's rule [Johnson, 1954] for minimizing the maximum completion time (the makespan) in the 2-machine flow shop. Then certainly, the condition (3.9) is satisfied. It is not hard to find out whether $TC(\pi^*) \leqslant TC(\pi)$; if so, then π is dominated by π^*. Note that, if J_j appears before J_k in π^*, while we have derived in Section 3.2.3 that $J_k \rightarrow J_j$, then we still can eliminate the node associated with π if the conditions (3.8) and (3.9) are satisfied.

The third rule looks for a job $J_k \in \pi$ such that $p_{1j} \leqslant p_{1k}$ and $p_{2j} \leqslant p_{2k}$. Thus, π can be written as $\pi = \rho_1 J_i \rho_2 J_j$, where ρ_1 and ρ_2 are subsequences. If we let $\pi^* = \rho_1 J_j \rho_2 J_k$, then the condition (3.8) for the dominance of π by π^* is satisfied. This is stated in the following lemma.

Lemma 3.1. *If we have* $p_{1j} \leqslant p_{1k}$ *and* $p_{2j} \leqslant p_{2k}$, *then* $TC(\rho_1 J_j \rho_2 J_k) \leqslant TC(\rho_1 J_k \rho_2 J_j)$.

Proof. We have $C_1(\rho_1 J_j) = C_1(\rho_1 J_k) + p_{1j} - p_{1j} \leqslant C_1(\rho_1 J_k)$; this implies

$$C_2(\rho_1 J_j) \leqslant C_2(\rho_2 J_k) + p_{2j} - p_{2k}. \tag{3.10}$$

Furthermore, we have $C_1(\rho_1 J_j \rho_l J_l) = C_1(\rho_1 J_k \rho_l J_l) + p_{1j} - p_{1k} \leqslant C_1(\rho_1 J_k \rho_l J_l)$ for every job $J_l \in \rho_2$, and hence that

$$C_2(\rho_1 J_j \rho_l J_l) \leqslant C_2(\rho_1 J_k \rho_l J_l), \qquad \text{for every } J_l \in \rho_2, \tag{3.11}$$

where ρ_l denotes the jobs of subsequence ρ_2 that are scheduled before J_l. In

addition, we have $C_1(\rho_1 J_j \rho_2 J_k) = C_1(\rho_1 J_k \rho_2 J_j)$. Because of this and since $C_2(\rho_1 J_j \rho_2) \leq C_2(\rho_1 J_k \rho_2)$, we have

$$C_2(\rho_1 J_j \rho_2 J_k) \leq C_2(\rho_1 J_k \rho_2 J_j) + p_{2k} - p_{2j}. \tag{3.12}$$

Totaling all completion times by use of the expressions (3.10), (3.11), and (3.12) yields the desired result. □

As can be seen from (3.12), there is no guarantee beforehand that the condition (3.9) is satisfied as well. It has yet to be verified if this is the case; only then is $\pi = \rho_1 J_k \rho_2 J_j$ dominated by $\pi^* = \rho_1 J_j \rho_2 J_k$. We may discard the node associated with π even if some of the precedence relations obtained in Section 3.2.3 are violated in the sequence π^*. In that case, we have $TC(\rho_1 J_k \rho_2 J_j \rho_3) \geq TC(\rho_1 J_j \rho_2 J_k \rho_3) > UB$, where $\rho_1 J_k \rho_2 J_j \rho_3$ and $\rho_1 J_j \rho_2 J_k \rho_3$ are complete schedules.

Conway et al. [1967] claim that there is an optimal solution in which J_j precedes J_k if $p_{1j} \leq p_{1k}$ and $p_{2j} \leq p_{2k}$. As can be seen from the expression (3.12), this cannot be established by the interchange argument used in the proof of Lemma 3.1. Szwarc [1983] shows the claim to be faulty by a counterexample.

Under a more stringent condition, however, we deduce the following result, which can be used to generate a priori precedence constraints.

THEOREM 3.2. *If for J_j and J_k it holds that $p_{2j} = p_{2k}$ and $p_{1j} \leq p_{1k}$, then there is an optimal permutation in which J_j precedes J_k.*

PROOF. We have to show that under these conditions any subsequence of the type $\rho_1 J_j \rho_2 J_k$ is dominated by $\rho_1 J_k \rho_2 J_j$ in terms of the conditions (3.8) and (3.9). The condition (3.8) is satisfied as can be seen from Lemma 3.1. Since $p_{2j} = p_{2k}$, the expression (3.12) reduces to $C_2(\rho_1 J_j \rho_2 J_k) \leq C_2(\rho_1 J_k \rho_2 J_j)$, which implies that $C_2(\pi^*) \leq C_2(\pi)$; hence, the condition (3.9) is satisfied, too. □

Of course, if for J_j and J_k we have $p_{2j} = p_{2k}$ and $p_{1j} = p_{1k}$, we allow either $J_j \rightarrow J_k$ or $J_k \rightarrow J_j$ in order to avoid the inconsistency to have both $J_j \rightarrow J_k$ and $J_k \rightarrow J_j$. Note that the combination of the precedence relations from Theorem 3.2 and the precedence relations generated as described in Section 3.2.3 cannot result in inconsistencies.

3.4. THE ALGORITHM

Before starting the actual branch-and-bound procedure, we do some preprocessing in order to find an upper bound, to derive precedence constraints, and to accelerate the calculations in a node of the tree. To obtain an upper bound, say, UB, we begin with a random permutation and we try to improve its sum of the job completion times by local interchanges. This procedure turned out to be robust, providing us with satisfactory initial upper bounds.

In addition, we approximate the search over the $O(n^2)$ points as described in Section 3.2.1 by a search over 21 points. Therefore, we store the 21 sequences that solve the problems (L_c) with $c = x/20$, $x = 0, 1, \ldots, 20$, respectively. This

search works sufficiently well due to the flatness of the function $L:c \rightarrow L(c)$ around the optimum. The storage implies a significant reduction in lower bound computation time, since we have to sort the jobs for each of these values of c only in the preprocessing phase; it takes then only linear time to compute $L(c)$ in a node of the tree.

In a similar fashion, we store the maximum perturbation values Δ_j^+ and Δ_j^- for each J_j $(j = 1, \ldots, n)$, which are computed as described in Section 3.2.2. These values depend on the set of unscheduled jobs and should actually be computed in each node of the tree. Although they are likely to increase if we go down the search tree, the loss in strength was more than compensated for by the reduction in computation time. The storage reduces the cost of lower bound strengthening in a node of the tree from $O(n^2)$ to $O(n)$ time.

Precedence constraints are only derived from the solution of problem (L_{c^*}), where c^* is the best choice among the 21 values for c. The completion times on both machines can easily be computed from (3.3), requiring linear time, albeit we can alternatively put $C_{2j} \leftarrow C_{2j} + \min_{1 \leqslant k \leqslant n} p_{1k}$ for each J_j $(j = 1, \ldots, n)$, since the second machine is surely idle until $\min_{1 \leqslant k \leqslant n} p_{1k}$. For problem (L_{c^*}), we try to derive precedence constraints as described in Section 3.2.3. For that purpose, we introduce an $n \times n$ matrix B with elements $b_{jk} = (L_c(j,k))$ and $b_{jj} = 0$. It is necessary to store this matrix, since new precedence constraints may be derived if we find a better upper bound.

	IGNALL and SCHRAGE ALGORITHM			PROPOSED ALGORITHM	
data set	max. # active nodes	total # nodes	time sec	total # nodes	time sec
10.1	5	53	0.9	9	1.5
10.2	13	84	0.9	10	1.3
10.3	18	152	1.0	14	1.5
10.4	117	728	3.1	57	1.9
10.5	135	957	3.9	169	2.7
15.1	1462	13718	93.0	693	9.5
15.2	2097	11156	116.9	388	7.4
15.3	1721	17712	142.4	603	9.7
15.4	676	2946	18.6	169	5.0
15.5	4280	35442	958.8	380	6.0
20.1	5213	(98.81%)	336.7	963	19.0
20.2	6411	(95.28%)	281.0	9235	95.4
20.3	5266	(97.12%)	182.2	1282	21.7
20.4	8909	(90.43%)	490.0	8846	102.6
20.5	8184	(96.72%)	422.4	4913	56.3

TABLE 3.1. Computational results on a VAX-780 computer.

The Ignall and Schrage algorithm follows a *best bound* strategy. For each of the

new nodes the corresponding lower bound is calculated and, if this lower bound is smaller than the current upper bound, this new node is inserted in a list of *active nodes*. This list is sorted in order of non-decreasing lower bounds. The node on top of this list is chosen to branch from. A significant advantage of such a list is that it facilitates dominance checking. However, in the worst case, the size of this list is exponential in the number of jobs. Computational experiments made it clear to us that this dominance checking was only advantageous for instances with n up to 10.

In contrast to the Ignall and Schrage procedure, we use an *active node* strategy. This means that we generate descendant nodes, of which there are at most $n-l$, for only one non-discarded node at level l. These descendant nodes are stored in a separate list and sorted according to a branching rule. We then branch from the node on the top of this list. Such a procedure requires only $O(n^2)$ space, since at each level l we have a list of at most $n-l$ jobs. The only thing that remains to explain is the branching rule. The new nodes that add some job J_j without unscheduled predecessors to an initial sequence π are sorted in non-decreasing order of $\Sigma_{J_k \notin \{\pi\}} b_{kj}$. This sum is supposed to reflect some notion of 'costs' if we schedule J_j before the other unscheduled jobs.

Both algorithms were coded in C, implemented on a VAX-780 computer, and tested on problems with 10, 15 and 20 jobs. The processing times for each job were taken from the uniform distribution [1,10], as Kohler and Steiglitz [1975] did in carrying out their experiments. Table 3.1 presents the results. The entries in the column 'maximum number of active nodes' give an indication of the space required by the Ignall and Schrage algorithm. The new algorithm outperforms the Ignall and Schrage procedure, although in case $n = 10$ it is sometimes slower. The main reason for this lies in the preprocessing phase.

As to the Ignall and Schrage algorithm with 20 jobs, computation was terminated after 10000 nodes. An entry within brackets represents the ratio in percentage upon termination between the lower bound of the first node in the list and the current upper bound.

Although the presented approach shows a significant improvement with respect to the Ignall and Schrage algorithm, the $F2||\Sigma C_j$ problem remains difficult to solve. It appeared from additional experiments that major difficulties are encountered for instances beyond 25 jobs.

3.5. Extensions

3.5.1. The $F2||\Sigma w_j C_j$ problem

Most of the results obtained here carry over to the more general $F2||\Sigma w_j C_j$ problem. In this problem, each J_j has some weight w_j, which expresses its importance relative to other jobs. By performing an analysis along the lines of Section 3.2, we find that the resulting linear ordering problem is solvable in polynomial time in case that either $\lambda_j = 0$ for each j, or that $\lambda_j = w_j$ for each j, or that $\lambda_j = w_j/2$ for each j ($j = 1, \ldots, n$). For this last choice of λ the weights of the linear ordering problem are in product form.

3.5.2. The $F \,|\,|\, \Sigma C_j$ problem

A similar analysis can be performed for the general problem with m machines if only permutation schedules are allowed. Although non-permutation schedules should be considered as well, optimization is usually confined to the set of permutation schedules. If the capacity and availability constraints are assumed to be implicitly present, then the permutation scheduling problem can be formulated as follows: determine completion times C_{ij} $(i = 1, \ldots, m,\ j = 1, \ldots, n)$ that minimize

$$\sum_{j=1}^{n} C_{mj}$$

subject to

$$C_{i+1,j} \geqslant C_{ij} + p_{i+1,j}, \quad \text{for } i = 2, \ldots, m,\ j = 1, \ldots, n. \tag{3.13}$$

The conditions (3.13) state the precedence relations between each pair of consecutive operations of the same job. If we introduce Lagrangian multipliers $\lambda_{ij} \geqslant 0$ $(i = 1, \ldots, m-1, j = 1, \ldots, n)$ to dualize the constraints (3.13), then the Lagrangian problem, referred to as problem (L_λ), is to minimize

$$\sum_{j=1}^{n} \left[\lambda_{1j} C_{1j} + \sum_{i=1}^{m-1} (\lambda_{i+1,j} - \lambda_{ij}) C_{i+1,j} + (1 - \lambda_{m-1,j}) C_{mj} \right] + \sum_{i=1}^{m-1} \sum_{j=1}^{n} \lambda_{ij} p_{i+1,j}.$$

Let $L(\lambda)$ be the optimal value of this problem. Clearly, we must ensure that

$$0 \leqslant \lambda_{1j} \leqslant \lambda_{2j} \leqslant \cdots \leqslant \lambda_{m-1,j} \leqslant 1, \quad \text{for } j = 1, \ldots, n, \tag{3.14}$$

in order to avoid that $L(\lambda) = -\infty$. For given Lagrangian multipliers that satisfy these requirements, the Lagrangian problem decomposes into m single-machine problems, each of which is easily solvable by Smith's rule. In parallel to the 2-machine case, the requirement to solve the Lagrangian problem over all permutation schedules transforms the Lagrangian problem into a linear ordering problem. The following theorem gives a sufficient condition for solving this linear ordering problem in polynomial time.

THEOREM 3.3. *The problem of finding a permutation schedule that solves the Lagrangian problem* (L_λ) *is solvable in polynomial time if* $\lambda_{ij} = \alpha_i$ *for* $i = 1, \ldots, m-1$, $j = 1, \ldots, n$, *with* $0 \leqslant \alpha_1 \leqslant \cdots \leqslant \alpha_{m-1} \leqslant 1$; *in this case, the problem is solved by sequencing the jobs in order of non-decreasing values* $\alpha_1 p_{1j} + \sum_{i=1}^{m-2} (\alpha_{i+1} - \alpha_i) p_{i+1,j} + (1 - \alpha_{m-1}) p_{mj}$.

PROOF. For this specific choice of the Lagrangian multipliers, the weights of the linear ordering problem are in product form. □

Let $L(\alpha_1, \ldots, \alpha_{m-1})$ denote the value $L(\lambda)$ with $\lambda_{ij} = \alpha_i$, for $i = 1, \ldots, m-1$, $j = 1, \ldots, n$. The restricted Lagrangian dual problem for the m-machine case is then to maximize

$$L(\alpha_1, \ldots, \alpha_{m-1})$$

subject to

$$0 \leqslant \alpha_1 \leqslant \cdots \leqslant \alpha_{m-1} \leqslant 1.$$

This problem is solvable in polynomial time by use of the ellipsoid algorithm (see Theorem 1.6). Since the ellipsoid algorithm is very slow in practice, it is better to consider an approximation problem for the Lagrangian dual problem. Since the Lagrangian problem is solvable in polynomial time, an ascent direction method can be easily developed; it will be similar to the one for the Lagrangian dual problem of the $1|prec|\Sigma w_j C_j$ problem (see Chapter 2).

Bansal [1977] extends the Ignall and Schrage lower bounds in a straightforward fashion to the m-machine case. A series of m relaxed versions of the original problem are considered, of which the ith version ($i = 1, \ldots, m$) is of the following type: minimize ΣC_{mj} if all machines but M_i are assumed to have infinite capacity. Clearly, the ith such problem is solved by sequencing the jobs in order of non-decreasing values p_{ij}. Bearing in mind that M_i cannot start processing before time $t = \Sigma_{h=1}^{i-1} \min_{1 \leqslant j \leqslant n} p_{hj}$, we compute the completion times C_{ij} ($j = 1, \ldots, n$) as

$$C_{mj} = C_{ij} + \sum_{h=i+1}^{m} p_{hj}, \qquad \text{for } j = 1, \ldots, n;$$

$\Sigma_{j=1}^{n} C_{mj}$ is then a lower bound on the optimal objective value.

It is easy to verify that the value $L(1,1,\ldots,1)$ concurs with Bansal's first lower bound, $L(0,1,\ldots,1)$ with the second, and so on; $L(0,0,\ldots,0)$, finally, concurs with Bansal's mth lower bound. Hence, the lower bound produced by an ascent direction method is at least as good as Bansal's best lower bound.

4

Parallel-Machine Scheduling

We consider the $R \mid \mid C_{\max}$ problem in this chapter. We present a 0-1 linear programming formulation for it, and subsequently dualize a set of constraints to obtain a Lagrangian problem with the integrality property; i.e., the optimal solution value of the linear programming relaxation equals the optimal solution value of the Lagrangian dual problem (see Corollary 1.1). The Lagrangian problem offers nonetheless attractive opportunities for the design of both an optimization algorithm and an approximation algorithm. The optimization algorithm solves large problems within reasonable time limits. The approximation algorithm is based upon a novel concept for iterative local search, where the search direction is guided by Lagrangian multipliers.

The organization of this chapter is as follows. In Section 4.1, we describe the problem in detail and give an overview of the literature for this problem. In Section 4.2, we formulate the $R \mid \mid C_{\max}$ problem as an integer linear program, examine the Lagrangian dual problem, and develop a quick ascent direction algorithm. In Section 4.3, we present the approximation algorithm. A complete description of the branch-and-bound algorithm is given in Section 4.4. Some computational results are presented in Section 4.5. Conclusions are given in Section 4.6.

4.1. Introduction

We first recall the specification of the $R \mid \mid C_{\max}$ problem. There are m parallel machines available for processing a set of n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each of these machines can handle at most one job at a time. The processing of job J_j ($j = 1, \ldots, n$) on machine M_i ($i = 1, \ldots, m$) requires a positive time p_{ij}. We may assume that these processing times are integral. Each job has to be scheduled on one of the machines and has to be processed without interruption. A *schedule* is an assignment of each of the jobs to exactly one machine. The

length of the schedule, also referred to as the *makespan*, is the maximum job completion time; by definition, the makespan is also equal to the maximum machine completion time. The objective is to find a schedule of minimum length.

The $R \mid \mid C_{max}$ problem has a range of potential applications. It arises in the context of computer system scheduling, where the machines are processors of a distributed computing environment with varying capabilities across the tasks. Other applications are found in the area of flexible manufacturing systems. For instance, a cluster of parallel machines may form a single or bottleneck stage in the production process. The problem also occurs in the context of machine load balancing, where machines have to be equipped with the appropriate tools for the jobs assigned to them. If production follows a cyclic pattern, and if the *system set-up time* (the time to load the machine with the appropriate tools) is costly relative to production time, then an obvious objective is to minimize the cycle time; note that cycle time minimization is equivalent to throughput maximization. Berrada and Stecke [1986] consider such a problem with limited capacities of the machines' tool magazines.

In Chapter 1, we have shown that the $R \mid \mid C_{max}$ problem is already $\mathcal{NP}$-hard in case of two identical machines. The traditional problem is then to balance solution quality with running time. An optimal solution may only be found at the expense of an exponential amount of computation time; a polynomial-time algorithm cannot be guaranteed to produce the optimal solution.

Two attempts have been made to solve the $R \mid \mid C_{max}$ problem to optimality. Stern [1976] presents a branch-and-bound algorithm; Horowitz and Sahni [1976] develop a dynamic programming procedure. In either case, no computational results are reported.

Much research effort has been invested in the development of approximation algorithms with a guaranteed accuracy. Ibarra and Kim [1977] and Davis and Jaffe [1981] propose various approximation algorithms with worst-case performance ratios that increase with the number of machines. For *fixed m* (i.e., the number of machines is specified as part of the problem type and not of the problem instance), Horowitz and Sahni [1976] give a fully polynomial approximation scheme with time and space complexity $O(nm(nm/(\rho-1))^{m-1})$. A *polynomial approximation scheme* is a family of algorithms that contains for any $\rho > 1$ a ρ-approximation algorithm with a running time that is bounded by a polynomial in the problem size; this running time may depend on ρ. A family of algorithms is called a *fully polynomial approximation scheme* if it contains for any $\rho > 1$ a ρ-approximation algorithm for which the running time is bounded by a polynomial in the problem size as well as in $1/(\rho-1)$.

Potts [1985B] presents a 2-approximation algorithm. Its time requirement is polynomial only for fixed m; its space requirement, however, is polynomial in m. For the 2-machine case, Potts improves the worst-case ratio to $(1+\sqrt{5})/2$. The algorithm is a two-phase procedure. In the first phase, linear programming is used to assign at least $n-m+1$ jobs; in the second phase, complete enumeration is used to schedule the remaining jobs. Using Potts' algorithm as the basis, Lenstra, Shmoys, and Tardos [1987] present a 2-approximation algorithm that is polynomial in m. They also present a polynomial approximation scheme for a

fixed number of machines, requiring space bounded by a polynomial in the problem size and $\log(1/(\rho-1))$. In addition, they prove a notable negative result: unless $\mathcal{P} = \mathcal{NP}$, no polynomial ρ-approximation algorithm exists for any $\rho < \frac{3}{2}$.

Two papers consider $R \mid \mid C_{\max}$ from an empirical point of view. De and Morton [1981] present several hybrid list scheduling algorithms and perform a large-scale computational testing. Our computational experiments exhibit, however, that their algorithms produce poor results. Hariri and Potts [1990] propose several two-phase heuristics that proceed in the spirit of Potts' 2-approximation algorithm. The first phase is identical: linear programming is used to schedule at least $n-m+1$ jobs. The second phase proceeds differently: a heuristic is used as a substitute for complete enumeration to schedule the remaining jobs. Note that Potts' 2-approximation algorithm dominates such two-phase heuristics in terms of quality but not in terms of speed. Hariri and Potts also consider several constructive heuristics, using them in conjunction with iterative local improvement procedures.

In spite of the considerable attention that the $R \mid \mid C_{\max}$ problem has received, there is still a lack of practical algorithms and computational insight. We address this issue here. We are concerned with methods that solve $R \mid \mid C_{\max}$ satisfactorily from a practical standpoint. We develop an exact algorithm and an approximation algorithm; both are based on Lagrangian relaxation and duality.

4.2. Minimizing makespan and its dual problem

In this section, we present an ascent direction method for the Lagrangian dual problem of $R \mid \mid C_{\max}$. We will also show that the search for a good approximate solution for the Lagrangian dual problem can almost be integrated with the search for a good approximate solution for the primal problem. First, we give a 0-1 linear programming formulation.

Evidently, there is an optimal solution in which the jobs are processed without delay. In addition, the ordering of the jobs on the machines is irrelevant for the length of the schedule. We are therefore actually looking for an *assignment* of jobs to machines. Accordingly, we introduce assignment variables x_{ij} $(i = 1, \ldots, m,\ j = 1, \ldots, n)$ that take the value 1 if J_j is scheduled on M_i, and 0 otherwise. If we let C_i denote the completion time of machine M_i, then we have $C_i = \sum_{j=1}^{n} p_{ij}x_{ij}$. The maximum value of the machine completion times, denoted by $C_{\max}$, is then the length of the schedule.

The $R \mid \mid C_{\max}$ problem, hereafter referred to as problem (P), is to determine values x_{ij} that minimize

$$C_{\max} \tag{P}$$

subject to

$$\sum_{j=1}^{n} p_{ij}x_{ij} \leqslant C_{\max}, \qquad \text{for } i = 1, \ldots, m, \tag{4.1}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad \text{for } j = 1, \ldots, n, \tag{4.2}$$

$$x_{ij} \in \{0,1\}, \qquad \text{for } i = 1,\ldots,m,\ j = 1,\ldots,n. \tag{4.3}$$

The conditions (4.1) ensure that the completion time of each machine is less than or equal to the length of the schedule; the conditions (4.2) guarantee that each job is assigned. The conditions (4.3) ensure that each job is scheduled on *exactly one* machine, thereby precluding preemption. If we replace the integrality constraints (4.3) with the weaker conditions $x_{ij} \geqslant 0$ $(i = 1,\ldots,m,\ j = 1,\ldots,n)$, then we obtain the *linear programming relaxation* ($\bar{\text{P}}$); this problem is solvable in polynomial time. Note the close resemblance between the $R \mid \mid C_{\max}$ problem and the generalized assignment problem (see Section 1.3.2).

When considering Lagrangian relaxation, we may be hesitant to dualize the conditions (4.1), since the resulting Lagrangian problem possesses the integrality property (see Corollary 1.1). Nonetheless, we choose to do so for two good reasons. First, a quick ascent direction method produces good approximate solutions for the associated Lagrangian dual problem. Second, for each vector of Lagrangian multipliers, we get a feasible solution for the $R \mid \mid C_{\max}$ problem.

The Lagrangian problem can be obtained by dualizing the constraints (4.1), and then by simplifying the objective function through normalization of the Lagrangian multipliers. However, the Lagrangian problem is easier to obtain by the so-called technique of *surrogate relaxation*. The central idea for this type of relaxation is to replace a set of nasty constraints with a single condition that is a weighted aggregation of these constraints. We aggregate the conditions (4.1). We introduce a vector of multipliers $\lambda = (\lambda_1,\ldots,\lambda_m) \geqslant 0$ with $\lambda_i > 0$ for at least one i $(i = 1,\ldots,m)$, and replace the conditions (4.1) with

$$\sum_{i=1}^{m} \lambda_i \sum_{j=1}^{n} p_{ij} x_{ij} \leqslant \sum_{i=1}^{m} \lambda_i C_{\max}, \tag{4.1a}$$

or, equivalently,

$$C_{\max} \geqslant \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_i p_{ij} x_{ij} \Big/ \sum_{i=1}^{m} \lambda_i. \tag{4.1b}$$

The surrogate relaxation problem, referred to as problem (L_λ), is then to determine $L(\lambda)$, which is the minimum of

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_i p_{ij} x_{ij} \Big/ \sum_{i=1}^{m} \lambda_i \tag{L_λ}$$

subject to

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad \text{for } j = 1,\ldots,n, \tag{4.2}$$

$$x_{ij} \in \{0,1\}, \qquad \text{for } i = 1,\ldots,m,\ j = 1,\ldots,n. \tag{4.3}$$

It is a matter of writing out to verify that the Lagrangian problem obtained by dualizing the constraints (4.1) boils down to exactly the same problem. Therefore, we refer to the above problem as the Lagrangian problem. It is due to the special structure of this problem that the surrogate relaxation problem and the

Lagrangian relaxation problem coincide. Generally, these problems are different. In theory, the best surrogate bound is at least as good as the best Lagrangian bound; in practice, the former is much harder to obtain. Greenberg and Pierskalla [1970] and Karwan and Rardin [1979] compare both relaxation methods.

Along the lines of Section 1.3, we make now some observations concerning the structure, the properties, and the solution of the Lagrangian problem. In the remainder, we let $v(\cdot)$ denote the optimal solution value of problem $(\cdot)$.

OBSERVATION 4.1. Problem (L_λ) provides a lower bound on v(P), since any solution that satisfies (4.1) also satisfies (4.1b) (but not necessarily vice versa). We have therefore that $L(\lambda) \leqslant v(P)$ for any vector $\lambda = (\lambda_1, \ldots, \lambda_m) \geqslant 0$ of Lagrangian multipliers with $\lambda_i > 0$ for at least one i ($i = 1, \ldots, m$).

OBSERVATION 4.2. Problem (L_λ) is solvable in $O(nm)$ time by assigning each job J_j to a machine M_h for which $\lambda_h p_{hj} = \min_{1 \leqslant i \leqslant m} \lambda_i p_{ij}$. Ties may be settled arbitrarily.

Note that $L(\lambda) = \Sigma_{j=1}^{n} \min_{1 \leqslant i \leqslant m} \lambda_i p_{ij} / \Sigma_{i=1}^{m} \lambda_i$. We refer to $\lambda_i p_{ij}$ as the *dual processing time* of J_j on M_i. The conditions (4.3) of the Lagrangian relaxation problem can be replaced with the conditions $x_{ij} \geqslant 0$ ($i = 1, \ldots, m, j = 1, \ldots, n$) without affecting the optimal value $L(\lambda)$. Hence, problem (L_λ) has the integrality property, since it can be solved as a linear programming problem.

OBSERVATION 4.3. Any solution to (L_λ) is also a feasible solution to the primal problem (P), for any vector $\lambda = (\lambda_1, \ldots, \lambda_m) \geqslant 0$ of Lagrangian multipliers with $\lambda_i > 0$ for at least one i ($i = 1, \ldots, m$).

The constraints (4.2) and (4.3) enforce the assignment of each job to exactly one machine. For a specific optimal solution of problem (L_λ), let $C_i(\lambda)$ denote the completion time of M_i. The approximate solution value is then $C_{\max}(\lambda) = \max_{1 \leqslant i \leqslant m} C_i(\lambda)$. The way we settle ties when solving problem (L_λ) affects $C_{\max}(\lambda)$.

OBSERVATION 4.4. The objective value $L(\lambda)$ is a *convex* combination of the machine completion times. This implies that $\min_{1 \leqslant i \leqslant m} C_i(\lambda) \leqslant L(\lambda) \leqslant \max_{1 \leqslant i \leqslant m} C_i(\lambda)$.

Consider the following instance of the $R \mid \mid C_{\max}$ problem, where eight jobs are to be scheduled on three machines with the processing times given in Table 4.1. We also used this instance in Chapter 1. Let $\lambda = (1,1,1)$ be the vector of Lagrangian multipliers. The Lagrangian problem (L_λ) is solved by assigning each job to the machine with the smallest processing time for it. The resulting schedule is represented by the Gantt chart of Figure 4.1. The initial choice $\lambda = (1,1,1)$ gives an elementary lower bound: it is the sum of the minimum processing times divided by the number of machines. The lower bound is $L(\lambda) = 18\frac{1}{3}$; the upper bound is $C_{\max}(\lambda) = 33$.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
M_1	6	3	10	12	11	14	8	6
M_2	10	∞	15	6	6	11	14	7
M_3	11	9	14	14	∞	10	10	9

TABLE 4.1. Processing time matrix.

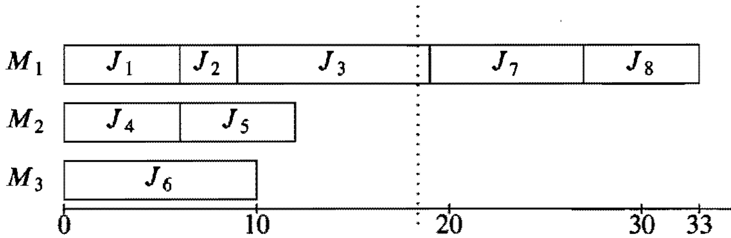


FIGURE 4.1. Gantt chart for $\lambda = (1,1,1)$; the dotted line indicates the lower bound $L(\lambda)$.

The best Lagrangian lower bound is found by solving the Lagrangian dual problem, referred to as problem (D). It is defined as

$$v(\mathrm{D}) = \max\{L(\lambda) \mid \lambda \geqslant 0\}. \tag{D}$$

In the remainder, we let λ^* denote the vector of optimal Lagrangian multipliers.

OBSERVATION 4.5. Since (L_λ) possesses the integrality property, we have $v(\bar{\mathrm{P}}) = v(\mathrm{D})$: the Lagrangian dual yields the same lower bound as the linear programming relaxation $(\bar{\mathrm{P}})$ (see Corollary 1.1).

This result is also derived in the following direct way. Geoffrion [1974] points out that it is possible to take the dual of a linear programming problem with respect to only a portion of the constraints. We assert that doing so for problem $(\bar{\mathrm{P}})$ with respect to the conditions (4.1) yields exactly the Lagrangian dual problem (D), since the conditions (4.3) in problem (L_λ) can be replaced with $x_{ij} \geqslant 0$ $(i = 1, \ldots, m, j = 1, \ldots, n)$.

Like the Lagrangian objective functions we examined in the previous chapters, the function $L : \lambda \rightarrow L(\lambda)$ is continuous in λ and everywhere differentiable except at the points where the Lagrangian problem (L_λ) has multiple optimal solutions. Unlike those other functions, the function L is not piecewise linear and not concave; this is because the term $\Sigma_{i=1}^{m} \lambda_i$ appears in the denominator of the Lagrangian objective function. However, we can still develop an ascent direction algorithm for approximating the optimal solution of problem (D). Some effort is required to find the primitive directional derivatives. The Lagrangian problem here apparently does not belong to the class of Lagrangian problems for which

we have shown that the primitive directional derivatives reduce to the dualized constraints (cf. Section 1.3.3.3). We will show, however, that the primitive directional derivatives for this particular Lagrangian objective function also reduce to simple expressions. Using the primitive directional derivatives, we also show that the shape of L between the points of non-differentiability does not matter: at any λ not being a point of non-differentiability, we can travel along a primitive direction to a point of non-differentiability where the Lagrangian function value is no worse than $L(\lambda)$. For the Lagrangian problem (D), this means that the optimization over all $\lambda \geqslant 0$ can be reduced to the optimization over all $\lambda \geqslant 0$ that correspond to points of non-differentiability. For the ascent direction procedure, we will therefore invariably compute step sizes that take us from one point of non-differentiability to another.

The ascent direction method for approximating the optimal solution of problem (D) is similar to the ascent direction method we described for the generalized assignment problem (see Section 1.3.3.2). First, we derive the primitive directional derivatives. Let $l_i^+(\lambda)$ be the primitive directional derivative for increasing λ_i; let $l_i^-(\lambda)$ be the primitive directional derivative for decreasing λ_i. From among the optimal solutions for problem (L_λ), let $x(i)^+$ be a solution with least jobs assigned to M_i, and let $x(i)^-$ be a solution with most jobs assigned to M_i, for $i = 1, \ldots, m$. To get an $x(i)^+$, each job J_j with $\lambda_i p_{ij}$ minimal and with $\lambda_h p_{hj} = \lambda_i p_{ij}$ for some $M_h \neq M_i$ is not assigned to M_i; all other ties are settled arbitrarily. To get an $x(i)^-$, each J_j with $\lambda_i p_{ij}$ minimal is assigned to M_i; all other ties are settled arbitrarily. Let $C_i^+(\lambda)$ be the completion time of M_i for such an $x(i)^+$; let $C_i^-(\lambda)$ be the completion time of M_i for such an $x(i)^-$. Let $\mathcal{J}_i^+(\lambda)$ denote the set of jobs on M_i for such an $x(i)^+$; let $\mathcal{J}_i^-(\lambda)$ denote the set of jobs on M_i for such an $x(i)^-$.

Recall that, at a higher level, the primitive directional derivatives are defined as

$$l_i^+(\lambda) = \lim_{\epsilon \downarrow 0} \frac{L(\lambda_1, \ldots, \lambda_i + \epsilon, \ldots, \lambda_m) - L(\lambda_1, \ldots, \lambda_m)}{\epsilon},$$

and

$$l_i^-(\lambda) = \lim_{\epsilon \downarrow 0} \frac{L(\lambda_1, \ldots, \lambda_i - \epsilon, \ldots, \lambda_m) - L(\lambda_1, \ldots, \lambda_m)}{\epsilon},$$

for $i = 1, \ldots, m$. For any h ($h = 1, \ldots, m$), let $\bar{\lambda} = (\lambda_1, \ldots, \lambda_h + \epsilon, \ldots, \lambda_m)$ with $\epsilon > 0$. We choose $\epsilon > 0$ sufficiently small to ensure that $x(h)^+$ remains optimal for problem ($L_{\bar{\lambda}}$); such an ϵ exists (see Theorem 1.7). For a specific $x(h)^+$, we have therefore that $C_i(\bar{\lambda}) = C_i(\lambda)$ for each i ($i = 1, \ldots, m$), and that $C_h(\lambda) = C_h^+(\lambda)$. Hence, we have

$$L(\bar{\lambda}) = \frac{\sum_{i=1}^{m} \bar{\lambda}_i C_i(\bar{\lambda})}{\sum_{i=1}^{m} \bar{\lambda}_i} = \frac{\epsilon C_h^+(\lambda) + \sum_{i=1}^{m} \lambda_i C_i(\lambda)}{\epsilon + \sum_{i=1}^{m} \lambda_i}$$

$$= \frac{\epsilon C_h^+(\lambda) + \dfrac{\sum_{i=1}^m \lambda_i C_i(\lambda)}{\sum_{i=1}^m \lambda_i} \sum_{i=1}^m \lambda_i}{\epsilon + \sum_{i=1}^m \lambda_i} = \frac{\epsilon C_h^+(\lambda) + L(\lambda) \sum_{i=1}^m \lambda_i}{\epsilon + \sum_{i=1}^m \lambda_i}.$$

This gives that

$$L(\bar{\lambda}) - L(\lambda) = \epsilon [C_h^+(\lambda) - L(\lambda)] / (\epsilon + \sum_{i=1}^m \lambda_i).$$

Using this, we obtain for the primitive directional derivative that

$$l_h^+(\lambda) = [C_h^+(\lambda) - L(\lambda)] / \sum_{i=1}^m \lambda_i.$$

In a similar fashion, we get that

$$l_h^-(\lambda) = [L(\lambda) - C_h^-(\lambda)] / \sum_{i=1}^m \lambda_i.$$

If $C_h^+(\lambda) > L(\lambda)$, then machine M_h is *overloaded*; maintaining the parallel with the generalized assignment problem, we say that $L(\lambda)$ is the *virtual* capacity of M_h. Increasing λ_h is then an ascent direction: we will obtain an improved Lagrangian objective value by moving along this direction. If $C_h^-(\lambda) < L(\lambda)$, then machine M_h is called *underloaded*; decreasing λ_h is an ascent direction.

Now we show that the shape of L between the points of non-differentiability does not matter. Suppose λ is not a point of non-differentiability: the Lagrangian problem (L_λ) has a single optimum. For $h = 1, \ldots, m$, let Δ_h^+ be the step size for increasing λ_h to reach the nearest point of non-differentiability, and let Δ_h^- be the step size for decreasing λ_h to reach the nearest point of non-differentiability. For $h = 1, \ldots, m$, let $\lambda(h)^+ = (\lambda_1, \ldots, \lambda_h + \Delta_h^+, \ldots, \lambda_m)$, and let $\lambda(h)^- = (\lambda_1, \ldots, \lambda_h - \Delta_h^-, \ldots, \lambda_m)$. Using the derivation of the primitive directional derivatives, we have

$$L(\lambda(h)^+) - L(\lambda) = \Delta_h^+ [C_h(\lambda) - L(\lambda)] / (\Delta_h^+ + \sum_{i=1}^m \lambda_i),$$

and

$$L(\lambda(h)^-) - L(\lambda) = \Delta_h^- [L(\lambda) - C_h(\lambda)] / (-\Delta_h^- + \sum_{i=1}^m \lambda_i).$$

Since $L(\lambda)$ is a convex combination of the machine completion times $C_1(\lambda), \ldots, C_m(\lambda)$, we have $L(\lambda(h)^+) \geqslant L(\lambda)$ for at least one h, or $L(\lambda(h)^-) \geqslant L(\lambda)$ for at least one h. Hence, for problem (D), we can restrict ourselves to the optimization over all $\lambda \geqslant 0$ corresponding to points of non-differentiability.

If we find an ascent direction, then we travel along this direction to the nearest

point where the associated primitive directional derivative changes. The required step size is easily determined. Suppose $l_h^+(\lambda) > 0$: M_h is overloaded. Increasing λ_h makes M_h less attractive to schedule jobs on. Eventually, we reach the first point where some J_j currently scheduled on M_h can equally well be scheduled on some other machine M_g; moving on beyond this point, we enforce the removal of J_j from M_h. The step size Δ to reach this point is the smallest positive value for which $(\lambda_h + \Delta)p_{hj} = \lambda_g p_{gj}$ for some J_j on M_h and some M_g $(M_g \neq M_h)$; hence, it is computed as

$$\Delta = -\lambda_h + \min_{1 \leq i \leq m,\, i \neq h,\, J_j \in \mathcal{J}_h^+(\lambda)} \lambda_i p_{ij} / p_{hj}.$$

Accordingly, we get $\bar{\lambda} = (\lambda_1, \ldots, \lambda_h + \Delta, \ldots, \lambda_m)$; the increment to the objective value is $L(\bar{\lambda}) - L(\lambda) = \Delta[C_h^+(\lambda) - L(\lambda)]/(\Delta + \Sigma_{i=1}^m \lambda_i) > 0$. Furthermore, we move J_j from M_h to M_g, and examine whether increasing $\bar{\lambda}_h$ is also an ascent direction.

Now, suppose $l_h^-(\lambda) > 0$: M_h is underloaded. Decreasing λ_h makes M_h more attractive. Eventually, we reach the first point where some J_j on M_g $(M_g \neq M_i)$ can equally well be scheduled on M_h; moving on beyond this point will force J_j to go to M_h. The required step size Δ is the smallest positive value for which $(\lambda_h - \Delta)p_{hj} = \lambda_g p_{gj}$, for some J_j scheduled on some M_g; it is computed as

$$\Delta = \lambda_h - \min_{1 \leq i \leq m,\, i \neq h,\, J_j \in \mathcal{J} - \mathcal{J}_h^-(\lambda)} \lambda_i p_{ij} / p_{hj}.$$

Accordingly, we get $\bar{\lambda} = (\lambda_1, \ldots, \lambda_h - \Delta, \ldots, \lambda_m)$, and the increment to the objective value is $L(\bar{\lambda}) - L(\lambda) = \Delta[L(\lambda) - C_h^-(\lambda)]/(-\Delta + \Sigma_{i=1}^m \lambda_i) > 0$; we move J_j to M_h, and examine whether decreasing $\bar{\lambda}_h$ is also an ascent direction.

If the ascent direction method is started in some $\lambda > 0$, then the ascent direction can never reach a boundary point where $\lambda_i = 0$ for some i $(i = 1, \ldots, m)$. Also, we must have $\lambda_i^* > 0$ for each i $(i = 1, \ldots, m)$; if $\lambda_i^* = 0$, then increasing λ_i^* is an ascent direction, thereby contradicting its optimality. Termination of the ascent direction method happens therefore at some $\bar{\lambda}$ where all primitive directional derivatives exist. At such a $\bar{\lambda}$, we have

$$l_i^+(\bar{\lambda}) \leq 0, \text{ and } l_i^-(\bar{\lambda}) \leq 0, \qquad \text{for } i = 1, \ldots, m,$$

or equivalently,

$$\sum_{J_j \in \mathcal{J}_i^+(\bar{\lambda})} p_{ij} \leq L(\bar{\lambda}) \leq \sum_{J_j \in \mathcal{J}_i^-(\bar{\lambda})} p_{ij}, \qquad \text{for } i = 1, \ldots, m.$$

The identification of the ascent direction and the computation of the step size can be implemented in different ways. We have freedom concerning the choice of the initial vector, and, for each iteration, the choice of the ascent direction. Since $\lambda_i^* > 0$ for each i $(i = 1, \ldots, m)$, we best start with a positive vector. Moreover, since the Lagrangian multipliers are normalized values, we can fix one multiplier a priori without running the risk of missing the optimum. The choice of the ascent direction affects the upper bounds that we get as by-products: for machine load balancing, it may be better to choose the direction of steepest ascent. Nonetheless, we give below a step-wise description of a rudimentary version, stripped from most of such considerations.

ASCENT DIRECTION ALGORITHM FOR PROBLEM (D)

Step 0. For $h = 1, \ldots, m$, set $\lambda_h \leftarrow 1$. Solve problem (L_λ), settling ties arbitrarily. Determine $L(\lambda)$.

Step 1. For $h = 1, \ldots, m$, do the following:

(a) While $C_h^+(\lambda) > L(\lambda)$, compute

$$\Delta = -\lambda_h + \min_{1 \leqslant i \leqslant m,\, i \neq h,\, J_j \in \mathcal{J}_h^+(\lambda)} \lambda_i p_{ij} / p_{hj},$$

set $\lambda_j \leftarrow \lambda_j + \Delta$, and update $L(\lambda)$ and $C_h^+(\lambda)$.

(b) While $C_h^-(\lambda) < L(\lambda)$, compute

$$\Delta = \lambda_h - \min_{1 \leqslant i \leqslant m,\, i \neq h,\, J_j \in \mathcal{J} - \mathcal{J}_h^-(\lambda)} \lambda_i p_{ij} / p_{hj},$$

set $\lambda_j \leftarrow \lambda_j - \Delta$, and update $L(\lambda)$ and $C_h^-(\lambda)$.

Step 2. Stop if no ascent direction was identified; otherwise, go to Step 1.

Let us reconsider our example and the solution of (L_λ) with $\lambda = (1,1,1)$. Machine M_1 is overloaded. The step size to remove some job from M_1 is $\Delta = \frac{1}{6}$: increasing λ_1 by $\frac{1}{6}$ allows us to move J_8 to M_2. We get $\lambda = (\frac{7}{6},1,1)$, a schedule with makespan 27, and $L(\lambda) = 19.1$ (see Figure 4.2; the dotted line indicates the virtual capacity of the machines).

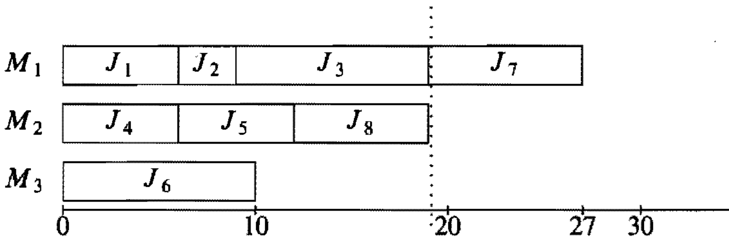


FIGURE 4.2. Gantt chart for $\lambda = (\frac{7}{6},1,1)$.

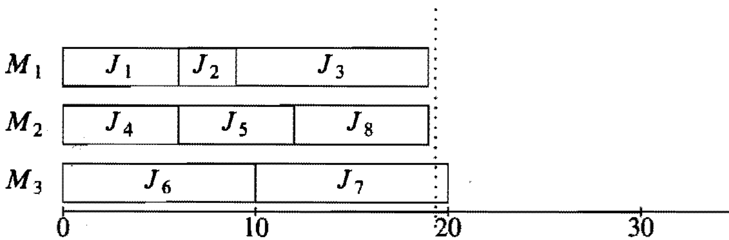


FIGURE 4.3. Gantt chart for $\lambda = (\frac{5}{4},1,1)$.

Machine M_1 remains overloaded; we increase λ_1 to $\frac{5}{4}$, and subsequently move J_7 to M_3. We get $\lambda = (\frac{5}{4},1,1)$, a schedule with makespan 20, and $L(\lambda) = 19.3$ (see Figure 4.3). Since all processing times are integral, the optimal makespan is integral as well. Hence, we have found an optimal primal solution. However, an

ascent direction still exists: M_2 is underloaded. If we decrease λ_2 by $\frac{1}{11}$, then J_6 goes to M_2. We obtain $\lambda = (\frac{5}{4}, \frac{10}{11}, 1)$ and $L(\lambda) = 19\frac{44}{139}$. At this point, no primitive ascent direction exists anymore: the ascent direction method is terminated at $\bar{\lambda} = (\frac{5}{4}, \frac{10}{11}, 1)$.

The vector $\bar{\lambda}$ is not optimal for the Lagrangian dual problem, i.e., $\bar{\lambda} \neq \lambda^*$. For λ^*, we have that $v(\bar{P}) = L(\lambda^*)$ (Observation 4.5), and that the complementary slackness conditions hold (Corollary 1.1). These conditions can be shown to imply that

$$x_{ij} > 0 \Leftrightarrow \lambda_i^* p_{ij} \text{ minimal, for } i = 1, \ldots, m, j = 1, \ldots, n.$$

Considering Figure 4.3, we would obtain a feasible solution with $v(\bar{P}) = L(\bar{\lambda})$ that satisfies the complementary slackness relations if and only if we could split J_6 over M_3 and M_2 and J_7 over M_3 and M_1 in order to process them before time $L(\bar{\lambda})$. This is not possible.

We now discuss Potts' 2-approximation algorithm and its relation to the ascent direction algorithm. For an arbitrary number of machines, the first phase of Potts' algorithm is to solve the linear programming relaxation $(\bar{P})$. The solution of $(\bar{P})$ shows at least $n - m + 1$ jobs each assigned to exactly one machine, and at most $m - 1$ jobs split over two or more machines. The jobs assigned to exactly one machine are retained as a partial schedule. The split jobs are assigned so as to minimize the makespan, given the partial schedule. Since $v(\bar{P}) \leqslant v(P)$, the length of the partial schedule is no more than $v(P)$. The scheduling of the split jobs proceeds by complete enumeration; this adds at most $v(P)$ to the length of the partial schedule. Hence, the resulting schedule has a makespan at most twice the optimal makespan. Since $(\bar{P})$ is solvable in polynomial time and complete enumeration for at most $m - 1$ split jobs requires $O(m^m)$ time, the procedure is polynomial for fixed m.

Consider now an optimal solution of problem $(L_{\bar{\lambda}})$. Since $\bar{\lambda}$ is the vector upon termination of the ascent direction method, we have

$$\sum_{J_j \in \mathcal{J}_i^+(\bar{\lambda})} p_{ij} \leqslant L(\bar{\lambda}) \leqslant \sum_{J_j \in \mathcal{J}_i^-(\bar{\lambda})} p_{ij}, \quad \text{for } i = 1, \ldots, m.$$

Exploiting these termination conditions, we point out a 2-approximation algorithm that proceeds entirely in the spirit of Potts' 2-approximation algorithm. In the first phase, we assign each $J_j \in \mathcal{J}_i^+(\bar{\lambda})$ to M_i, thus obtaining a partial schedule with length no more than $v(P)$. The remaining jobs, contained in the set $\mathcal{J} - \cup_{i=1}^m \mathcal{J}_i^+(\bar{\lambda})$, have ties concerning their minimal dual processing times. In the second phase, we assign these jobs by complete enumeration so as to minimize the makespan, given the partial schedule; this adds at most $v(P)$ to the length of the partial schedule. Hence, the resulting schedule has makespan no more than $2v(P)$.

In fact, we get also a schedule with worst-case ratio 2 with fewer jobs to assign in the second phase as follows. Let $\mathcal{J}_i^0(\bar{\lambda}) \subseteq \mathcal{J}_i^-(\bar{\lambda}) - \mathcal{J}_i^+(\bar{\lambda})$ $(i = 1, \ldots, m)$ be mutually disjoint subsets of jobs such that

$$\sum_{J_j \in \mathcal{J}_i^+(\bar{\lambda}) \cup \mathcal{J}_i^0(\bar{\lambda})} p_{ij} \leqslant L(\bar{\lambda}), \quad \text{for each } i = 1, \ldots, m;$$

hence, $\mathcal{J}_i^0(\bar{\lambda})$ contains only jobs with ties concerning their minimal dual processing time. In the first phase, we assign each $J_j \in \mathcal{J}_i^+(\bar{\lambda}) \cup \mathcal{J}_i^0(\bar{\lambda})$ to M_i; in the second phase, we assign the remaining jobs. The sets $\mathcal{J}_i^0(\bar{\lambda})$ should be chosen so as to minimize the number of jobs left for the second phase. In general, we cannot bound the number of jobs to be assigned in the second phase by a polynomial in m. However, if $\bar{\lambda} = \lambda^*$, then this procedure is exactly Potts' 2-approximation algorithm; since the complementary slackness relations hold for $\bar{\lambda} = \lambda^*$, we can choose the sets $\mathcal{J}_i^0(\bar{\lambda})$ in such a way that no more than $m-1$ jobs remain for the second phase.

For the special case $m = 2$, we have $v(\bar{\mathrm{P}}) = L(\bar{\lambda})$. Since the Lagrangian multipliers represent normalized values, only $m-1$ multipliers need in general to be involved to find λ^*: only one multiplier is involved for the case $m = 2$. The termination conditions at $\bar{\lambda}$ are then sufficient for optimality (see Theorem 1.7). For $m = 2$, problem $(\bar{\mathrm{P}})$ is solvable in $O(n)$ time [Gonzalez, Lawler, and Sahni, 1990]. Moreover, there is at most one split job. Considering the ascent direction algorithm, we observe that the solution generated by Potts' 2-approximation algorithm concurs with the best upper bound found when solving problem (D) by use of the ascent direction procedure.

4.3. Duality-based heuristic search

The principle of Potts' 2-approximation algorithm and specifically the termination conditions of the ascent direction algorithm give rise to the idea that a near-optimal solution for the Lagrangian dual problem induces a near-optimal solution for the primal problem. In this respect, we need a scheme that generates a series of promising Lagrangian multipliers. The example suggests that the ascent direction method, perhaps with some minor adjustments, is such a scheme. The ascent direction method, however, is too restrictive for our purpose. Computational experiments show that is usually terminated after only a small number of iterations. We need a scheme that allows us to browse quickly through many near-optimal solutions for problem (D). The approximation algorithm differs therefore from the ascent direction method on two counts.

First, the machine with the largest overload is always selected for multiplier adjustment. From a primal point of view, this is an obvious choice: one of the jobs on this machine must be removed in order to reduce the machine completion time that induces the current makespan. Second, we make the step size larger than necessary to enforce such a removal: this avoids early termination. Specifically, we move to the *second* point where the primitive directional derivative changes. Let machine M_h be the machine with the largest overload in the solution of problem (L_λ); hence, we have $C_{\max}(\lambda) \leqslant C_h^+(\lambda)$. Then we compute

$$\Delta = -\lambda_h + \min2_{1 \leqslant i \leqslant m, J_j \in \mathcal{J}_h^+(\lambda)} \lambda_i p_{ij} / p_{hj},$$

where min2 denotes the second minimum of these values. If we put $\bar{\lambda} = (\lambda_1, \ldots, \lambda_h + \Delta, \ldots, \lambda_m)$, then we enforce the move of some J_k from M_h to

some M_g, and that another job on M_h can equally well be scheduled on some other machine. Nonetheless, this second job is kept on M_h. The next step is to compute the new makespan $C_{\max}(\bar{\lambda})$, and the machine with the largest overload; this machine is determined by computing $\max_{1 \leqslant i \leqslant m} C_i^+(\bar{\lambda})$. We have no guarantee that the rescheduling of J_k induces an improved schedule: we can have either $C_{\max}(\bar{\lambda}) \leqslant C_{\max}(\lambda)$ or $C_{\max}(\bar{\lambda}) > C_{\max}(\lambda)$. The latter occurs if $C_g(\bar{\lambda}) = C_g(\lambda) + p_{gk} > C_{\max}(\lambda)$. Hence, the approximation algorithm is equipped with a mechanism that accepts deteriorations of the makespan. We repeat this process for the machine with the largest load, and store the best solution on the way. We put an upper bound on the number of iterations, since this procedure does not have any convergence properties. Below we give a stepwise description of the algorithm; *maxiter* is a prespecified maximum number of iterations and *UB* is the currently best solution value.

APPROXIMATION ALGORITHM

Step 0. Put $\lambda \leftarrow (1, \ldots, 1)$, $t \leftarrow 1$. Solve (L_λ), settling ties arbitrarily. Let $UB \leftarrow C_{\max}(\lambda)$, and store the schedule.

Step 1. Determine M_h with the largest overload: $C_h^+(\lambda) \geqslant C_i^+(\lambda)$ for each i $(i = 1, \ldots, m)$. Compute Δ, and identify a job J_k and a machine M_g such that $\lambda_g p_{gk} / p_{hk} = \min_{1 \leqslant i \leqslant m,\, i \neq h,\, J_j \in \mathcal{J}_h^+(\lambda)} \lambda_i p_{ij} / p_{hj}$. Put $t \leftarrow t + 1$.

Step 2. Put $\lambda \leftarrow (\lambda_1, \ldots, \lambda_h + \Delta, \ldots, \lambda_m)$, $C_h(\lambda) \leftarrow C_h(\lambda) - p_{hk}$, $C_g(\lambda) \leftarrow C_g(\lambda) + p_{gk}$. If $C_{\max}(\lambda) < UB$, then $UB \leftarrow C_{\max}(\lambda)$, and store the schedule. If $t < maxiter$, then go to Step 1; if not, then stop.

We call the approximation algorithm described above the *duality-based approximation algorithm*, and the particular strategy employed as *duality-based heuristic search*. For the example, the approximation algorithm goes through the same steps as described in Section 4.2.

Many heuristic search strategies are applicable to the parallel machine scheduling problem (see Section 1.2.3). An *iterative local improvement procedure* is a local-search type of algorithm, which can be designed as follows for the $R \mid \mid C_{\max}$ problem. Let σ be some arbitrary schedule and let σ_{jk} be the schedule obtained from σ by swapping J_j and J_k $(j \neq k)$. We define the so-called *single pairwise interchange neighborhood* for σ as the set N_σ containing the schedules σ_{jk} for all $j = 1, \ldots, n-1$, $k = j+1, \ldots, n$. Suppose M_h is such that $C_h(\sigma) = C_{\max}(\sigma)$, where $C_h(\sigma)$ and $C_{\max}(\sigma)$ denote the completion time of M_h and the maximum machine completion time in σ, respectively. Let (J_j, J_k) be a pair of jobs such that J_j is scheduled on M_h and J_k on some other machine M_g $(g \neq h)$ for which we have

$$C_g + p_{gj} - p_{gk} < C_h, \text{ and } C_h - p_{hj} + p_{hk} < C_h.$$

If we interchange J_j and J_k, that is, we put J_j on M_g and J_k on M_h, then we reduce the makespan. In other words, we have identified a schedule $\sigma_{jk} \in N_\sigma$ with $C_{\max}(\sigma_{jk}) < C_{\max}(\sigma)$. This process is repeated until no further improvement is found. As said before, the danger is to get stuck in a poor local optimum.

Simulated annealing and tabu search are techniques that try to avoid such an entrapment by allowing deteriorations of the objective value under certain circumstances. The willingness to accept deteriorations unconditionally distinguishes the duality-based search technique from simulated annealing, tabu search, and general iterative local improvement schemes.

Anticipating on the implementation and the evaluation of the duality-based approximation algorithm in Section 4.5, however, we will consider two versions of the algorithm. On the one hand, we evaluate the duality-based algorithm on its own; on the other hand, we evaluate the algorithm in conjunction with the iterative local improvement procedure we described. We only submitted the best solution to the improvement procedure. The duality-based algorithm in conjunction with the iterative local improvement procedure produces very good results. Apparently, the duality-based approximation algorithm finds an attractive initial solution for the iterative local improvement procedure.

4.4. The branch-and-bound algorithm

The first step in the branch-and-bound algorithm is to run the ascent direction method to approximate the optimal solution of problem (D). Upon termination, we have the vector $\bar{\lambda}=(\bar{\lambda}_1, \ldots, \bar{\lambda}_m)$ of Lagrangian multipliers. On the way, we store the best primal solution. We also use the duality-based approximation algorithm and the constructive heuristics presented by De and Morton [1980], Ibarra and Kim [1977], and Davis and Jaffe [1981] to find approximate solutions for problem (P). The implementation of these algorithms is described in Section 4.5. The vector $\bar{\lambda}$ plays an important role in the truncation of the search tree.

4.4.1. Initial reductions

The size of an instance may be reduced by a simple reduction test, which is common for linear programming theory. It can be conducted for any vector of Lagrangian multipliers, but success is most likely for λ^* and vectors close to it.

Theorem 4.3. *If for a given vector of multipliers $\lambda = (\lambda_1, \ldots, \lambda_m)$, we have for some J_k and M_h that*

$$(\lambda_h p_{hk} - \min_{1 \leq i \leq m} \lambda_i p_{ik}) / \sum_{i=1}^{m} \lambda_i > UB - L(\lambda) - 1,$$

where UB is a given upper bound on $v(\mathrm{P})$, then $x_{hk} = 0$ in any schedule with $C_{\max} < UB$, if such a schedule exists.

Proof. Suppose there is a schedule with makespan less than UB, and yet with J_k scheduled on M_h. Solving the Lagrangian relaxation problem (L_λ) under the additional constraint $x_{hk} = 1$ gives the lower bound LB with

$$\begin{aligned} LB &= (\lambda_h p_{hk} + \sum_{j=1; j \neq k}^{n} \min_{1 \leq i \leq m} \lambda_i p_{ij}) / \sum_{i=1}^{m} \lambda_i \\ &= [(\lambda_h p_{hk} - \min_{1 \leq i \leq m} \lambda_i p_{ik}) + \sum_{j=1}^{n} \min_{1 \leq i \leq m} \lambda_i p_{ij}] / \sum_{i=1}^{m} \lambda_i > UB - 1, \end{aligned}$$

which is a contradiction. □

4.4.2. The search tree

A node at level k of the search tree corresponds to a partial schedule with a specific assignment of $J_1, \ldots, J_k$. Each node at level k ($k = 1, \ldots, n-1$) has at most m descendant nodes: one node for the assignment of job J_{k+1} to each machine M_i, for $i = 1, \ldots, m$. The jobs and the machines will be reindexed in compliance with the branching rule we propose in the next subsection. The algorithm we use is of the 'depth-first' type. We employ an *active node* search: at each level, we consider only one node to branch from, thereby adding some job to the partial schedule. The nodes are branched from in order of increasing indices of the associated machines. We backtrack if we reach the bottom of the tree or if we can discard the active node.

4.4.3. Branching rule

The dual processing times $\bar{\lambda}_i p_{ij}$ ($i = 1, \ldots, m$, $j = 1, \ldots, n$) also serve to structure the search tree. We define $\gamma_j = \text{min2}_{1 \leqslant i \leqslant m} \bar{\lambda}_i p_{ij} - \min_{1 \leqslant i \leqslant m} \bar{\lambda}_i p_{ij}$, where min2 denotes the second minimum. In view of Theorem 4.3, a large value γ_j suggests that there exists an optimal solution with J_j scheduled on the machine with minimum dual processing time for it; we call this machine the *favorite* machine for J_j. We like to structure the search tree in such a way that we first explore the configurations with jobs with large γ_j assigned to their favorite machines. This is achieved by reindexing the jobs in order of non-increasing values γ_j and by reindexing the machines at each level k ($k = 1, \ldots, n-1$) in order of non-decreasing values $\bar{\lambda}_i p_{i,k+1}$ ($i = 1, \ldots, m$). We note that the first complete schedule encountered in the tree is an optimal solution for the Lagrangian problem ($L_{\bar{\lambda}}$).

Such a structure of the search tree has two advantages. First, for the optimal solution and good approximate solutions of the primal problem, most jobs are expected to have been assigned to their favorite machines. Second, if we find an improved upper bound, then most of the additional variable reductions are associated with the nodes of the still unexplored part of the search tree.

4.4.4. Discarding nodes

Here, we describe in detail the various rules to discard nodes. Computational experiments show, surprisingly enough, that even a quick ascent direction method is not worthwhile to be run in each node of the tree. We use therefore the vector $\bar{\lambda} = (\bar{\lambda}_1, \ldots, \bar{\lambda}_m)$ throughout the search tree. The reduction test and the following rules depend on $\bar{\lambda}$. The vector λ^* may therefore be more effective; it may be worthwhile to use a linear programming algorithm in the root of the tree to obtain λ^*. On the other hand, if $\bar{\lambda}$ is close to λ^*, then the additional effect will be negligible. Suppose the values z_{ij} ($i = 1, \ldots, m$, $j = 1, \ldots, k$) record the current partial schedule at level k of the tree. That is, $z_{ij} = 1$ if J_j has been assigned to M_i, and $z_{ij} = 0$ otherwise. Let $L(\bar{\lambda}, k)$ denote the optimal solution of problem ($L_{\bar{\lambda}}$) subject to $x_{ij} = z_{ij}$ for $i = 1, \ldots, m$, $j = 1, \ldots, k$. Then we have

$$L(\bar{\lambda}, k) = L(\bar{\lambda}) + \sum_{j=1}^{k} \left(\sum_{i=1}^{m} (\bar{\lambda}_i p_{ij} - \min_{1 \leqslant i \leqslant m} \bar{\lambda}_i p_{ij}) z_{ij} \right) / \sum_{i=1}^{m} \bar{\lambda}_i.$$

Note that $L(\bar{\lambda}, k) \geq L(\bar{\lambda})$. A node at level k that assigns J_k to machine M_h can be discarded if

$$(\bar{\lambda}_h p_{hk} - \min_{1 \leq i \leq m} \bar{\lambda}_i p_{ik}) / \sum_{i=1}^{m} \bar{\lambda}_i > UB - L(\bar{\lambda}, k-1) - 1. \qquad \text{(F1)}$$

This test requires constant time per node. In addition, the node can be discarded if

$$\sum_{j=1}^{k-1} p_{hj} z_{hj} + p_{hk} > UB - 1. \qquad \text{(F2)}$$

The third test tries to establish whether the current partial schedule is dominated by another partial schedule for the same k jobs. Suppose we have some job J_l $(1 \leq l \leq k-1)$ that is currently scheduled on M_i for which

$$p_{il} > p_{ik} \text{ and } p_{hl} < p_{hk}. \qquad \text{(F3)}$$

Interchanging J_l and J_k reduces the load of both M_i and M_h. The current partial schedule can then be discarded, since there is at least one optimal schedule with no such pair of jobs.

Conditions similar to (F2) apply to each job J_j $(j = k+1, \ldots, n)$. In case there is a job J_l $(k+1 \leq l \leq n)$ for which

$$\sum_{j=1}^{k} p_{ij} z_{ij} + p_{il} > UB - 1, \quad \text{for each } M_i, \, i = 1, \ldots, m, \qquad \text{(F4)}$$

we discard the node, too. Similarly, if the condition (F4) applies to some J_l $(k+1 \leq l \leq n)$ for all machines M_i $(i = 1, \ldots, m)$ but one, we can assign J_l to this machine. Subsequently, we can possibly carry out additional assignments; these, in turn, enhance the likelihood that the node is closed on account of (F1), (F2), (F3), or (F4).

In addition, we try to identify a machine M_h $(1 \leq h \leq m)$ for which

$$\sum_{j=1}^{l} p_{hj} z_{hj} + p_{hl} > UB - 1, \quad \text{for each } J_l, \, l = k+1, \ldots, n.$$

In this case, M_h is ignored for the assignment of any remaining job. Therefore, we discard the node if

$$\left[\sum_{j=1}^{k} \sum_{i=1, i \neq h}^{m} \bar{\lambda}_i p_{ij} z_{ij} + \sum_{j=k+1}^{n} \min_{1 \leq i \leq m, \, i \neq h} \bar{\lambda}_i p_{ij} \right] / \sum_{i=1; i \neq h}^{m} \bar{\lambda}_i > UB - 1.$$

4.5. Computational experiments

Both algorithms have been coded in the computer language C; the experiments were conducted on a Compaq-386/20 Personal Computer.

The algorithms were tested on a broad range of instances with n and m varying from 20 to 200 and from 2 to 20, respectively, giving rise to 80 combinations altogether. The processing times were generated from the uniform distribution [10,100]. For each combination of n and m we considered 10 instances.

$n\downarrow m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	1	2	0	0	0
50	0	0	0	0	1	5	-	-	-	0
60	0	0	0	0	1	-	-	-	-	1
80	0	0	0	2	-	-	-	-	-	-
100	0	0	0	3	-	-	-	-	-	-
200	0	1	0	-	-	-	-	-	-	-

TABLE 4.2. Number of unresolved problems out of 10 for each cell.

$n\downarrow m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	16	46	68	203	180	75	33	37	11	0
30	31	90	340	752	434	1440	784	145	4784	64
40	37	170	615	4488	10149	6786	23936	3800	192	342
50	59	171	1188	6133	16022	48202	-	-	-	5848
60	68	358	1127	12715	27942	-	-	-	-	10669
80	85	1232	3386	37110	-	-	-	-	-	-
100	132	2503	5198	28116	-	-	-	-	-	-
200	330	12245	14274	-	-	-	-	-	-	-

TABLE 4.3. Average number of nodes.

$n\downarrow$ $m\rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1	1	1	1	1	1	1	1	1	1
30	1	1	1	2	2	9	6	2	43	2
40	1	1	2	12	39	39	214	63	3	10
50	1	1	3	16	57	285	-	-	-	204
60	1	1	3	33	105	-	-	-	-	373
80	1	3	8	96	-	-	-	-	-	-
100	1	6	12	87	-	-	-	-	-	-
200	3	40	52	-	-	-	-	-	-	-

TABLE 4.4. Average computation time in seconds.

4.5.1. The branch-and-bound algorithm

For the branch-and-bound algorithm we put an upper bound of 100,000 nodes; computation for any instance was discontinued at this limit. In Table 4.2, we present for each combination the number of unresolved problems. An empty cell indicates that the branch-and-bound algorithm was not run; considering adjacent cells or initial computations, we expected that most of the instances would remain unresolved. Table 4.3 shows the average number of nodes explored. The average for a particular combination of n and m is computed by aggregating the number of nodes for each of the instances and dividing the sum by 10, the total number of instances for each combination. Unresolved instances contribute therefore 10,000 nodes each to the average number of nodes. Table 4.4 presents the average computation time for the branch-and-bound algorithm, including the running time for the heuristics and the duality-based approximation algorithm. The time spent on unresolved instances is included, too. The average computation time for a particular combination is computed in a similar fashion as the average number of nodes.

From a practical point of view, the instances with a few machines are easy. The effort required to solve a problem seems to increase more with the number of machines than with the number of jobs. Surprising exceptions are the instances with $m \geqslant 12$ and $n \leqslant 40$. Note that the 100,000-node limit for the branch-and-bound algorithm is arbitrary: it induces distinct *time* limits across the instances. For example, instances with $m = 20$ and $n = 50$ or 60 require about 10,000 nodes on the average; however, they require about 5 minutes of running time. Nonetheless, one can easily form some idea about the instances that are within reach of, say, one minute of computation time.

Significant deviations from the averages occur. For the combination $n = 30$ and $m = 15$, for example, a single instance accounts for the remarkably large number of nodes and large running time. It is also conceivable that the performance of the algorithm is enhanced by fine-tuning the algorithm to particular instances. For large values of n and m, for example, it may be worthwhile to use the ascent direction method in each node of the tree after all. Even then, however, such instances are not solvable within reasonable time limits.

4.5.2. The duality-based approximation algorithm

Implementing the duality-based approximation algorithm, we have put *maxiter* $= nm$. Note that cycling may occur. This happens, for instance, if J_j can be scheduled on both M_1 and M_2. If J_j is scheduled on M_1, then M_1 has the largest overload; if J_j is scheduled on M_2, then M_2 has the largest overload. In such a situation, J_j would oscillate between M_1 and M_2. The procedure is discontinued upon detection of this phenomenon.

The duality-based approximation algorithm was compared with the constructive heuristics of De and Morton [1980], Ibarra and Kim [1977], Davis and Jaffe [1981], and with our version of Potts' 2-approximation algorithm [Potts, 1985B] (see Section 4.2); the latter is easy to embed in the branch-and-bound algorithm. We have evaluated neither Potts' original version, nor the 2-approximation algorithm presented by Lenstra, Shmoys, and Tardos [1990], nor the two-phase

heuristics presented by Hariri and Potts [1990]. All these algorithms proceed in the same spirit; none is expected to outperform the others significantly in practice. In the remainder, when referring to Potts' 2-approximation algorithm, we are actually referring to our version of it. Recall that the versions are identical for $m=2$. The constructive heuristics display a very erroneous behavior. For instance, the De and Morton heuristic, taking the best result from 10 underlying heuristics, produces solutions with an average deviation from the best solution of 27%. We have therefore treated the constructive heuristics as a single algorithm by considering only the best schedule.

In Table 4.5, we present the average relative deviation for the best schedule generated by the constructive heuristics from the optimal solution, or if this is not available, from the best known solution. In the latter case, brackets have been placed around the figures. Table 4.6 shows the same information for the duality-based approximation algorithm.

$n \downarrow m \rightarrow$	2	3	4	5	6	8	10	12	15	20
20	2.9	8.0	6.8	12.8	19.0	20.3	7.6	22.6	8.9	5.4
30	2.2	6.3	8.2	18.0	18.8	25.5	22.5	23.1	14.6	13.0
40	3.0	6.5	10.8	14.6	13.3	(27.5)	(27.4)	25.6	28.6	19.0
50	2.0	7.2	12.0	12.1	(19.3)	(23.4)	(17.2)	(18.4)	(14.7)	32.8
60	1.4	6.0	10.3	10.5	(15.9)	(14.6)	(17.2)	(15.4)	(16.9)	(42.5)
80	1.8	4.6	8.4	(11.1)	(13.4)	(11.4)	(16.9)	(17.5)	(24.7)	(20.5)
100	2.8	3.3	7.2	(9.7)	(11.1)	(15.0)	(19.8)	(18.3)	(19.8)	(21.2)
200	0.7	(2.4)	3.9	(4.2)	(5.0)	(8.2)	(15.5)	(17.0)	(15.5)	(24.5)

TABLE 4.5. Average relative deviation for the constructive heuristics.

$n \downarrow m \rightarrow$	2	3	4	5	6	8	10	12	15	20
20	4.2	5.4	6.6	10.5	11.3	16.4	15.2	14.6	7.4	3.0
30	1.5	4.9	6.0	9.8	8.9	14.7	16.7	21.0	14.0	15.2
40	1.9	4.2	3.5	9.0	8.3	(10.0)	(19.0)	14.4	13.5	19.3
50	1.6	3.3	4.9	7.4	(6.5)	(8.3)	(4.1)	(1.9)	(2.5)	18.1
60	1.2	1.1	4.1	5.5	(5.0)	(4.0)	(1.8)	(1.0)	(1.8)	(24.3
80	1.4	2.3	2.9	(3.5)	(2.4)	(1.9)	(2.1)	(2.8)	(2.7)	(4.5)
100	2.3	2.3	2.4	(3.6)	(1.8)	(2.2)	(1.9)	(1.9)	(0.8)	(1.4)
200	0.4	(1.5)	1.1	(1.1)	(1.2)	(1.8)	(3.3)	(1.3)	(3.3)	(0.6)

TABLE 4.6. Average relative deviation for the duality-based approximation algorithm.

As a whole, the duality-based approximation algorithm performs much better than the constructive heuristics, which behave poorly. This certainly applies to

instances with a larger number of machines. The performance of the constructive heuristics is easily improved by submitting them to an iterative local improvement scheme. Therefore, the schedules generated by the constructive heuristics should merely be seen as initial solutions that serve as input for some iterative local improvement procedure.

Each schedule generated by the constructive heuristics was therefore subsequently submitted to the iterative local improvement procedure we described in Section 4.3. In contrast, only the best schedule generated by the duality-based approximation algorithm was submitted to the improvement procedure.

In Tables 4.7, 4.8, and 4.9, we present the results for the constructive heuristics, Potts' 2-approximation algorithm, and the duality-based approximation algorithm after local improvement, respectively. The sign '*' behind an entry in these tables indicates that the corresponding algorithm has the best average performance for the associated instances. Table 4.7 exhibits that the iterative local improvement technique is effective for the constructive heuristics in case of few machines or jobs. However, its effectiveness deteriorates with an increasing number of machines. Only two machines at a time are involved in the job interchanges. For large m, it is more difficult to find an attractive local neighborhood, even in case of multiple start solutions. Generally, the running time, which seems to be increasing with n, is modest: instances up to $n = 100$ require only one or two seconds; approximately 10 seconds of computation time are required for instances with $n = 200$. Because the job interchanges affect only two machines at a time, the number of machines hardly seems to play a role in the computation time.

Potts' 2-approximation algorithm was embedded in a branch-and-bound algorithm that differs on two points from the branch-and-bound algorithm described in Section 4.4. First, we omitted the dominance rule (F3); second, we initially put $UB = \infty$. The condition (F3) is useful for finding an optimal solution, but might eliminate good approximate solutions. That is why Potts' 2-approximation algorithm sometimes took more time than the optimization algorithm. Occasionally, more than $m-1$ jobs remained for the second phase. It is surprising that the final solution was rarely improved by the local improvement procedure, although it was applied to all jobs. The computational effort for the algorithm was modest and seemed to increase more with the number of machines than with the number of jobs. For instances up to $m = 12$, it was one or two seconds; for instances with $m = 15$ and $m = 20$, it was about 15 to 20 seconds. Instances with $n = 20$ and $m = 20$ were not run; for these instances, Potts' algorithm requires explicit enumeration of almost the entire state space.

As can be seen from the number of '*' signs in Table 4.9, the duality-based approximation algorithm has the best performance on the average. Note that the entries for $m = 2$ are identical for the duality-based algorithm and Potts' algorithm. In spite of their close relation, the duality-based approximation algorithm performs considerably better than Potts' algorithm.

Table 4.10 presents the number of times (out of 10) that the duality-based approximation algorithm produced the best or equally best solution. The algorithm performs remarkably well if m and n are large; apparently, these instances

$n \downarrow m \rightarrow$	2	3	4	5	6	8	10	12	15	20
20	0.0*	1.0*	3.0*	7.1	8.3*	10.4	4.9*	11.1	1.9*	5.1
30	0.1*	1.2*	3.1	4.4*	7.6	13.3*	15.0	17.4*	11.0	9.2
40	0.2*	1.3	1.7*	3.8*	7.9	(13.6)	(15.4)	17.8	22.2	14.7*
50	0.3*	1.1*	2.7*	5.2	(7.8)	(11.6)	(5.0)	(6.9)	(7.0)	20.4
60	0.2*	1.0	3.1	3.8	(5.9)	(4.8)	(2.5)	(7.4)	(10.6)	(32.8)
80	0.1*	0.9	2.3	(2.8)	(1.9)	(1.9)	(4.7)	(6.7)	(12.5)	(12.4)
100	2.5	0.7	1.9	(2.9)	(1.7)	(2.1)	(6.1)	(6.1)	(6.1)	(10.6)
200	0.2	(0.6)	1.1	(0.8)	(1.3)	(1.8)	(3.5)	(4.1)	(3.5)	(8.7)

TABLE 4.7. Average relative deviation for the constructive heuristics after iterative local improvement.

$n \downarrow m \rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1.7	3.0	5.4*	9.1	11.0	8.9*	10.8	14.2	4.8	-
30	0.2	2.7*	4.6	7.9	6.9	13.4	12.7*	22.0	10.4*	3.7*
40	0.4	2.1	2.8	5.3	8.7	(13.6)	(17.6)	19.5	20.6	15.1
50	0.4	1.6	3.3	5.6	(7.6)	(11.8)	(8.6)	(6.6)	(6.2)	21.6
60	0.3	2.6	2.8	5.1	(6.7)	(1.9)	(3.1)	(10.1)	(4.2)	(37.0)
80	0.1*	1.9	2.2	(5.7)	(4.0)	(3.4)	(9.5)	(7.2)	(10.3)	(12.4)
100	2.3*	1.7	1.9	(4.4)	(3.1)	(2.6)	(3.9)	(3.9)	(9.3)	(10.6)
200	0.1*	(0.8)	1.0	(1.7)	(2.4)	(3.9)	(5.0)	(6.3)	(5.0)	(14.7)

TABLE 4.8. Average relative deviation for Potts' 2-approximation algorithm after iterative local improvement.

$n \downarrow m \rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1.7	1.0*	5.4	5.4*	9.8	14.5	14.0	10.8*	7.4	3.0*
30	0.2	2.6	3.9	5.2	6.7*	14.2	15.0	19.7	11.3	14.8
40	0.4	1.0*	2.8	5.2	4.4*	(9.3)*	(15.2)*	13.9*	12.0*	16.4
50	0.4	1.5	2.3*	4.1*	(4.2)*	(7.2)*	(1.3)*	(0.0)*	(1.2)*	17.0*
60	0.3	0.5*	2.0*	2.7*	(3.5)	(1.0)*	(0.8)*	(0.8)*	(0.9)*	(22.8)*
80	0.1*	0.7*	1.0*	(1.9)*	(1.8)*	(0.6)*	(0.6)*	(2.2)*	(1.6)*	(4.5)*
100	2.3*	0.6*	1.1*	(2.2)*	(0.8)*	(0.9)*	(0.7)*	(0.7)*	(0.3)*	(0.7)*
200	0.1*	(0.5)*	0.7*	(0.3)*	(0.1)*	(0.8)*	(1.1)*	(0.4)*	(1.2)*	(0.0)*

TABLE 4.9. Average relative deviation for the duality-based approximation algorithm after iterative local improvement.

$n \downarrow$ $m \rightarrow$	2	3	4	5	6	8	10	12	15	20
20	1	7	5	7	4	4	3	6	7	9
30	7	4	4	4	7	6	4	7	4	4
40	5	6	3	3	8	9	4	7	9	5
50	7	4	5	7	7	6	6	10	8	9
60	4	8	7	6	8	4	5	9	8	9
80	2	5	7	6	9	6	8	6	9	7
100	9	4	5	6	5	6	8	8	9	9
200	6	7	6	7	9	7	8	8	8	10

TABLE 4.10. Number of times (out of 10) that the duality-based approximation algorithm performed at least as well as the other approximation algorithms.

are beyond the reach of the iterative local improvement procedure and Potts' 2-approximation algorithm. In a sense, the duality-based approximation algorithm and the branch-and-bound algorithm are supplementary: the latter is effective for instances for which the former performs not so well as the other approximation algorithms. The running time is about a factor of two more than the running time of the constructive heuristics and Potts' approximation algorithm, but it is comparable or less in the extreme combinations with $n = 200$ or $m = 20$.

4.6. Conclusions

The $R \mid \mid C_{\max}$ problem is a practical scheduling problem for which we have proposed a branch-and-bound algorithm and an approximation algorithm. The branch-and-bound algorithm solves large instances to optimality within reasonable time limits. The approximation algorithm is based upon a simple and intuitively appealing idea for local search: heuristic duality-based search in conjunction with iterative local improvement. For instances that are beyond the reach of an optimization algorithm, it produces very good results.

5

Common Due Date Scheduling

We consider here the single-machine problem of minimizing the sum of the deviations of the job completion times from a given common due date that is restrictively small, i.e., smaller than the sum of the processing times. This problem is known to be $\mathcal{NP}$-hard. Previous algorithms include a pseudo-polynomial algorithm solving instances up to 1000 jobs, and a branch-and-bound algorithm solving instances up to only 25 jobs. We apply Lagrangian relaxation to find new lower and upper bounds that coincide for virtually all instances with the number of jobs not too small. The crux is the 'logic' formulation of the problem: it can be formulated as an easy matching problem complicated by only one constraint.

This chapter is organized as follows. In Section 5.1, we introduce the problem. In Section 5.2, we review Emmons' matching algorithm [Emmons, 1987] to solve the unrestricted variant of the common due date problem. In Section 5.3, we develop a lower bound based upon Lagrangian relaxation for the restricted variant. In Section 5.4, we use the insight gained in Section 5.3 to develop a heuristic for the restricted variant. In Section 5.5, we describe some details of the branch-and-bound algorithm. Finally, in Section 5.6, we present some computational results.

5.1. Introduction

The just-in-time concept for manufacturing has induced a new type of machine scheduling problem in which both early and tardy completions of jobs are penalized. We consider the following single-machine scheduling problem that is associated with this concept.

A set of n independent jobs has to be scheduled on a single machine, which can handle no more than one job at a time. The machine is continuously available from time 0 onwards. Job J_j requires processing during a given uninterrupted time p_j and is ideally completed exactly on a given due date d_j. Without

loss of generality, we assume that the processing times and the due dates are integral. We assume furthermore that the jobs are indexed in order of non-increasing processing times. A *schedule* σ defines for each job J_j a completion time C_j, such that the jobs do not overlap in their execution. The earliness and tardiness of J_j are defined as $E_j = \max\{d_j - C_j, 0\}$ and $T_j = \max\{C_j - d_j, 0\}$, respectively. The just-in-time philosophy is reflected in the objective function

$$f(\sigma) = \sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j),$$

where the deviation of C_j from d_j is penalized by either α_j or β_j, depending on whether J_j is early or tardy, for $j = 1, \ldots, n$. For a review of problems with this type of objective function, we refer to Baker and Scudder [1990].

An important subclass contains problems with a due date d that is common to all jobs. Either the common due date is specified as part of the problem instance, or the common due date is a decision variable that has to be optimized simultaneously with the job sequence. As the first job may start later than time 0, the optimal schedule is identical for both problems unless the common due date d is restrictively small ($d < \Sigma_{j=1}^{n} p_j$). The first variant is therefore referred to as the restricted problem (i.e., d is fixed) and the second variant as the unrestricted problem (i.e., d is a variable).

We consider the restricted variant of the problem in which all earliness penalties are equal to α and all tardiness penalties are equal to β. Bagchi, Chang, and Sullivan [1987] propose a branch-and-bound approach for this problem, and Szwarc [1989] presents a branch-and-bound approach for the case that $\alpha = \beta$. These branch-and-bound algorithms are able to solve instances up to 25 jobs. Sundararaghavan and Ahmed [1984] present an approximation algorithm for the case $\alpha = \beta$ that shows a remarkably good performance from an empirical point of view. Hall, Kubiak, and Sethi [1991] and Hoogeveen and Van de Velde [1991A] establish the $\mathcal{NP}$-hardness of the problem, even if $\alpha = \beta$, thereby justifying the enumerative and approximative approaches. Furthermore, Hall et al. [1991] propose a pseudo-polynomial time algorithm running in $O(n\Sigma_{j=1}^{n} p_j)$ time and space, and provide computational results for instances up to 1000 jobs.

Their experiments, however, show that the space requirement rather than the time requirement limits the applicability of the algorithm. In general, there is always need for a branch-and-bound algorithm that solves instances for which the space requirement is prohibitive. We present a branch-and-bound algorithm that solves virtually *all* instances without branching. It is based upon new lower and upper bounds, which are computed in $O(n \log n)$ time. If these bounds do not concur, they can be refined by solving a subset-sum problem to optimality by a pseudo-polynomial algorithm. This can be done very fast, since the subset-sum problem in our application is of a considerably smaller dimension than the common due date problem. Hence, the branch-and-bound algorithm is more than competitive with the pseudo-polynomial algorithm for the common due date problem.

the property that the work processed before time d is minimal among all optimal schedules for the Lagrangian problem (L_λ). In the same fashion, the schedule $\sigma_\lambda^{\max}$ is defined as the optimal schedule for the Lagrangian problem (L_λ) with a maximal amount of work processed before time d, for $\lambda = 0, \ldots, n$. We define $W_\lambda^{\min}$ and $W_\lambda^{\max}$ as the amount of work processed before time d in $\sigma_\lambda^{\min}$ and $\sigma_\lambda^{\max}$, respectively. Straightforward calculations show that $\sigma_\lambda^{\min}$ is identical to $\sigma_{\lambda+1}^{\max}$ and that $W_\lambda^{\min} = W_{\lambda+1}^{\max}$. This implies that $L(\lambda)$ is a piecewise linear and concave function of λ. The breakpoints correspond to the integral values $\lambda = 1, \ldots, n$, and the gradient of the function between the integral breakpoints λ and $\lambda+1$ is equal to $W_\lambda^{\min} - d$, for $\lambda = 0, \ldots, n-1$. The Lagrangian dual problem is therefore solved by putting λ^* equal to the index λ for which $W_\lambda^{\max} \geqslant d > W_\lambda^{\min}$. Due to the indexing of the jobs, the theorem follows. □

Let σ^* be an optimal schedule for the Lagrangian dual problem. If $\lambda^* = 0$, then $\sigma^* = \sigma_0^{\min}$ is feasible for the original problem, and hence optimal. Note that this also implies that $d \geqslant p_1 + p_3 + \cdots + p_n$ if n is odd, and $d \geqslant p_1 + p_3 \cdots + p_{n-1}$ if n is even. This agrees with the observation by Bagchi et al. [1987] that the schedules $(J_1, J_3, \ldots, J_n, J_{n-1}, \ldots, J_2)$ and $(J_1, J_3, \ldots, J_{n-1}, J_n, \ldots, J_2)$ are optimal under the respective conditions.

In the remainder, we assume that $\lambda^* \geqslant 1$. Depending on whether $n - \lambda^*$ is odd or even, σ^* has the following structure. First, suppose $n - \lambda^*$ is odd. Then the jobs $J_1, \ldots, J_{\lambda^*-1}$ occupy the last $\lambda^* - 1$ positions in σ^*, the pair $\{J_{\lambda^*}, J_{\lambda^*+1}\}$ occupies the first early position and the λ^*th tardy position, the pair $\{J_{\lambda^*+2}, J_{\lambda^*+3}\}$ occupies the second early position and the (λ^*+1)th tardy position, and so on. Finally, the pair $\{J_{n-1}, J_n\}$ occupies the positions around the due date. Second, if $n - \lambda^*$ is even, then σ^* has the same structure, except that J_n is positioned between J_{n-2} and J_{n-1}, and is started somewhere in the interval $[d - p_n, d]$.

THEOREM 5.5. *If there exists a schedule σ^* that is optimal for the Lagrangian dual problem in which the first job is started at time 0, then the Lagrangian lower bound $L(\lambda^*)$ is tight and σ^* is an optimal schedule for the original problem.* □

If no such schedule σ^* exists, then there is a gap between the optimal value for the original problem and the Lagrangian lower bound. This lower bound, however, can be strengthened by solving the *modified* Lagrangian problem, which is to find a schedule that minimizes

$$\sum_{j=1}^{n} |C_j - d| + \lambda^*(W - d) + |W - d|.$$

Clearly, the modified Lagrangian problem yields a lower bound for the original problem if $\lambda^* \geqslant 1$.

THEOREM 5.6. *The modified Lagrangian problem is solved by a schedule from among the schedules that are optimal for the Lagrangian dual problem, for which $|W - d|$ is minimal.*

PROOF. Suppose that π is optimal for the modified Lagrangian problem, but not for the Lagrangian dual problem. We show that π can be transformed without additional cost into a schedule $\bar{\pi}$ that is optimal for the Lagrangian dual problem by conducting pairwise interchanges. Let π_t be the schedule after t interchanges; hence, $\pi_0 = \pi$, and $\pi_T = \bar{\pi}$, for some $T \geqslant 1$. Note that it is possible to specify a series of pairwise interchanges that lowers the Lagrangian cost $\Sigma_{j=1}^{n} |C_j - d| + \lambda^* (W - d)$ at every interchange. Consider two successive schedules π_t and π_{t+1}, and suppose that J_i and J_j with $p_i > p_j$ have been interchanged. The interchange must have decreased the Lagrangian cost by at least $p_i - p_j$, and may have increased the term $|W - d|$ by at most $p_i - p_j$. This observation implies that every interchange does not increase $\Sigma_{j=1}^{n} |C_j - d| + \lambda^*(W - d) + |W - d|$. Therefore, $\bar{\pi}$ must also be optimal. □

Let σ^* now be an optimal schedule for the modified Lagrangian problem. Suppose that the first job in σ^* is not started at time 0. First, suppose that the first job is started before time 0. We then shift σ^* to make it feasible. Shifting σ^* implies that some jobs or parts of jobs are transferred to the other side of the due date. Let Δ_j be the amount of the jth job that has been transferred. If $n - \lambda^*$ is odd, then shifting σ^* increases $\Sigma_{j=1}^{n} |C_j - d|$ by $\lambda^* + 1$ per unit of the first job that is transferred, by $\lambda^* + 3$ per unit of the second job that is transferred, and so on. As $W - d$ is weighted by λ^*, the cost of the schedule after shifting is equal to $L(\lambda^*) + \Delta_1 + 3\Delta_2 + \cdots$. If $n - \lambda^*$ is even, then a similar analysis shows that the cost of the schedule after shifting is equal to $L(\lambda^*) + 2\Delta_2 + 4\Delta_4 \cdots$.

Second, suppose that the first job is started after time 0. Shifting by transferring jobs or parts of jobs to the other side of the due date decreases $\Sigma_{j=1}^{n} |C_j - d|$ as long as the number of jobs started before the due date is not greater than the number of jobs completed after the due date plus one. This implies that no more than $\lceil \lambda^*/2 \rceil$ jobs are transferred. If $n - \lambda^*$ is odd, then shifting σ^* decreases $\Sigma_{j=1}^{n} |C_j - d|$ by $\lambda^* - 1$ per unit of the first job that is transferred, $\lambda^* - 3$ per unit of the second job that is transferred, and so on. As $W - d$ is weighted by λ^*, the cost of the schedule after shifting is equal to $L(\lambda^*) + \lambda^*(d - W) + \Delta_1 + 3\Delta_2 \cdots$, where W is the amount of work before d after shifting. If $n - \lambda^*$ is even, then a similar analysis shows that the cost of the scheduling after shifting is equal to $L(\lambda^*) + \lambda^*(d - W) + 2\Delta_2 + 4\Delta_3 \cdots$; W is the amount of work before d after shifting.

THEOREM 5.7. *The schedule obtained after shifting σ^* has cost equal to the strengthened lower bound if $\lambda^* = 1$ and the first job is started after time 0, or if $n - \lambda^*$ is odd and either J_{n-1} or J_n is executed at time d.* □

However, in order to determine that schedule, we have to solve the tie-breaking problem. We will deal with this problem in the next section.

The lower bound approach can be extended to the restricted variant of the problem with $\alpha \neq \beta$. Without loss of generality, we assume that α and β are integral and relatively prime. A similar analysis shows that the optimal value λ^*

can be determined as the value $\lambda^* \in \{1, \ldots, n\beta\}$ for which $W_{\lambda^*}^{\max} \geqslant d > W_{\lambda^*}^{\min}$. Theorem 5.6 still holds, but the strengthening of the bound is less meaningful, since the cost of transferring one unit of processing time of the first job to the other side of d has been increased from 1 to either α or β, in case $n - \lambda^*$ is odd.

5.4. A NEW UPPER BOUND FOR THE RESTRICTED VARIANT

We start with the case $\alpha = \beta$. The analysis in the previous section suggests to find an optimal schedule for the Lagrangian dual problem with minimal $|W - d|$. This requires the development of a tie-breaking rule in Emmons' matching algorithm that minimizes $|W - d|$. Such a schedule induces an approximate solution to the common due date problem. This schedule is provably optimal if $W = d$ or if the conditions of Theorem 5.7 are satisfied.

We show that the problem of minimizing $|W - d|$ boils down to solving the optimization version of the *subset-sum* problem. This problem will be defined below; it will henceforth be referred to as the subset-sum problem. Although this problem is $\mathcal{NP}$-hard in the ordinary sense [Garey and Johnson, 1979], the instances occurring in our application virtually always belong to an easy-to-solve subclass if n is not too small. In general, the subset-sum problem is solvable by a dynamic programming procedure that requires significantly less effort than the $O(n\Sigma_{j=1}^{n} p_j)$ time and space algorithm for the common due date problem.

The problem of minimizing $|W - d|$ can be transformed into an instance of the subset-sum problem in the following way. Define $a_j = p_{2j-2+\lambda^*} - p_{2j-1+\lambda^*}$, for $j = 1, \ldots, \lfloor (n - \lambda^* + 1)/2 \rfloor$, and define $D = d - W_{\lambda^*}^{\min}$. Note that all $a_j \geqslant 0$. Remove the values a_j that are equal to 0; let m be the number of remaining values a_j, and let $\mathcal{A}$ be the multiset that contains the values $a_1, \ldots, a_m$. First, suppose that $n - \lambda^*$ is odd. The problem of minimizing $|W - d|$ is then equivalent to determining a subset $A \subseteq \mathcal{A}$, whose sum is as close to D as possible.

Second, suppose that $n - \lambda^*$ is even. An optimal schedule for the Lagrangian dual problem is also optimal for the original problem if $W \in [d - p_n, d]$. Finding such a schedule is equivalent to determining a subset $A \subseteq \mathcal{A}$ whose sum falls in the interval $[D - p_n, D]$. If no such subset exists, then the goal is to find a subset $A \subseteq \mathcal{A}$ whose sum is as close as possible to either $D - p_n$ or D.

Given a subset $A \subseteq \mathcal{A}$ (optimal or approximate), we determine the corresponding schedule for the common due date problem in the following way. Start with $\sigma_{\lambda^*}^{\min}$. Interchange the jobs that correspond to $a_j \in A$ for $j = 1, \ldots, m$, thereby increasing the amount of work processed before d by a_j. Finally, shift the schedule to ensure that the first job is started at time 0.

We show now how to determine a suitable set A. We reindex the values a_i in order of non-decreasing values. For n not too small, the instances of the subset-sum problem virtually always possess the *divisibility* property.

DEFINITION 5.1. A set of integers $\{a_1, \ldots, a_m\}$, with $1 = a_1 \leqslant a_2 \leqslant \cdots \leqslant a_m$, is said to possess the *divisibility property* if for every value $D \in \{1, 2, \ldots, \Sigma_{i=1}^{j} a_i\}$ there exists a subset $A \in \{a_1, \ldots, a_j\}$, whose sum is equal to D.

THEOREM 5.8. *A multiset of integers* $\{a_1, \ldots, a_m\}$, *with* $1 = a_1 \leqslant a_2 \leqslant \cdots \leqslant a_m$, *possesses the divisibility property if and only if* $a_{j+1} \leqslant \Sigma_{i=1}^{j} a_j + 1$, *for* $j = 1, \ldots, n-1$. □

An intuitive reason explains why virtually all instances with n not too small have this property. Every a_j is equal to the difference in processing times between two successive jobs in the shortest processing time order. This implies that for randomly generated instances of the common due date problem the values a_j tend to be small if n is not too small. Note that $\Sigma_{j=1}^{n} a_j \leqslant \max_{1 \leqslant j \leqslant n} p_j$.

THEOREM 5.9. *If an instance of the subset-sum problem possesses the divisibility property, then Johnson's greedy algorithm for subset-sum* [Johnson, 1974] *solves this instance to optimality in* $O(m \log m)$ *time.* □

JOHNSON'S ALGORITHM

Step 0. Reindex the values a_j in order of non-increasing values.
Step 1. Select the largest remaining value a_j with $a_j \leqslant D$. If there is no such value, then stop.
Step 2. Put a_j in the subset; $D \leftarrow D - a_j$.
Step 3. If $D \geqslant a_1$ and if a_1 is not in the subset, then go to Step 1.

Johnson's algorithm always yields a subset whose sum is no more than D. This handicap is overcome by not only applying the algorithm with the value D but also with the value $\Sigma_{j=1}^{m} a_j - D$. Let A be the subset in the latter case for which an approximate schedule can be constructed as described above, with $A = \mathcal{Q} \setminus \bar{A}$.

If Johnson's algorithm does not yield a provably optimal solution, then we solve the instance to optimality by dynamic programming. This requires $O(mD)$ time and space. By this, we may improve both on the upper and lower bound. If the lower and the upper bound still do not coincide, then we need to apply branch-and-bound to solve the common due date problem to optimality.

The approximation algorithm described above can be adjusted in an obvious fashion to deal with the restricted variant of the common due date problem with $\alpha \neq \beta$.

5.5. BRANCH-AND-BOUND

We describe the branch-and-bound algorithm for the case $\alpha = \beta$. The first step in the algorithm is to solve the Lagrangian dual problem. If $\lambda^* = 0$, then $\sigma^* = \sigma_0^{\min}$ is an optimal solution for the common due date problem, and we are done. Otherwise, we determine upper bounds as described in Section 5.4; we also apply the heuristic presented by Sundararaghavan and Ahmed. If the lower and the best upper bound do not concur, then we solve the subset-sum problem to optimality by dynamic programming. If the bounds still do not concur, then we apply branch-and-bound. In the remainder, we assume that the jobs have been reindexed in order of non-increasing processing times.

There is an optimal schedule in which either the jobs are scheduled in the

interval $[0,\Sigma_{j=1}^{n} p_j]$, or d coincides with the start time or completion time of the job with the smallest processing time; see also Theorem 5.2. To cope with these possibilities, we need to design two search trees. We make use of the following observation. In any optimal schedule, the jobs completed before or at the common due date d are scheduled in order of non-increasing processing times; the jobs started at or after d are scheduled in order of non-decreasing processing times. In case the jobs are scheduled in the interval $[0, \Sigma_{j=1}^{n} p_j]$, there may be a job that is started before and finished after d; for this particular job, it holds that the early fr the tardy jobs have larger processing times. Due to this structure, optimal schedules are said to be V-shaped. This observation is easily verified by use of an interchange argument.

For the case that the jobs are scheduled in the interval $[0,\Sigma_{j=1}^{n} p_j]$, the search tree has the following form. A node at level j $(j = 1, \ldots, n)$ of the search tree corresponds to a partial schedule in which the completion times of the jobs $J_1, \ldots, J_j$ are fixed. Each node at level j has at most $(n-j)$ descendants. In the kth $(k = 1, \ldots, n-j)$ descendant, J_{j+k} is started before d; if $k \geqslant 2$, then the jobs $J_{j+1}, \ldots, J_{j+k-1}$ are to be completed after d. Given the partial schedule for $J_1, \ldots, J_j$, a partial schedule for $J_1, \ldots, J_{j+k}$ is easily computed.

For the case that d coincides with the start time or completion time of the job with the smallest processing time, i.e., J_n, the tree has the following form. A node at level j $(j = 1, \ldots, n)$ of the search tree corresponds to a partial schedule in which the completion times of the jobs $J_n, \ldots, J_{n-j+1}$ are fixed. Each node at level j has at most $(n-j)$ descendants. In the kth $(k = n-j, \ldots, 1)$ descendant, $J_{n-j-k+1}$ is started before d; if $k \geqslant 2$, then the jobs $J_{n-j}, \ldots, J_{n-j-k+2}$ are to be completed after d. Given the partial schedule for $J_n, \ldots, J_{n-j+1}$, a partial schedule for $J_n, \ldots, J_{n-j-k+1}$ can easily be computed.

Both trees are successively explored according to a 'depth-first' strategy. We employ an *active node* search: at each level we choose one node to branch from. In the tree for the case the jobs are scheduled in the interval $[0,\Sigma_{j=1}^{n} p_j]$, we consistently choose the node, whose job has the smallest remaining index; in the tree for the case d coincides with either the completion or start time of J_n, we choose the node, whose job has the largest remaining index.

A simple but powerful rule to restrict the growth of each search tree is the following. A node at level j $(j = 1, \ldots, n)$ corresponding to some J_k can be discarded if another node at the same level corresponding to some J_l with $p_k = p_l$ has already been considered. This rule obviously avoids duplication of schedules.

As far as lower bounding in the nodes of the tree is concerned, we only compute the lower bound $L(\lambda^*)$. Hence, we neither solve the modified Lagrangian dual problem nor compute additional upper bounds.

5.6. Computational results

The processing times were drawn from the uniform distribution [1, 100]. Computational experiments were performed with $d = \lfloor t\Sigma_{j=1}^{n} p_j \rfloor$ for $t =$ 0.1, 0.2, 0.3, 0.4, respectively, and with the number of jobs ranging from 10 to 1000. For each combination of n and t we generated 100 instances. The algorithm was coded in the computer language C; the experiments were conducted

on a Compaq-386/20 Personal Computer.

The results are shown in Table 5.1, the design of which reflects our three-phase approach. The third column '# $O(n \log n)$' shows the number of times (out of 100) that Johnson's subset-sum algorithm gave rise to a schedule with cost equal to the Lagrangian lower bound $L(\lambda^*)$; this is the number of times that the common due date problem was provably solved to optimality in $O(n \log n)$ time. The fourth column '# DP' shows how many of the remaining instances were provably solved to optimality by dynamic programming applied to subset-sum. The fifth column 'maximum # nodes' shows the maximum number of nodes that was needed for the branch-and-bound algorithm. The sixth column '# greedy optimal' shows the number of times that Johnson's algorithm induced an optimal schedule. The seventh column '# SA optimal' gives the same information for the approximation algorithm presented by Sundararaghavan and Ahmed. The last column '# LB tight' shows the number of times that the lower bound (strengthened or not) was equal to the optimal solution value.

We conclude that the common due date problem is easy to solve from a practical point of view. As pointed out in Section 5.4, a randomly generated instance with n not too small can be expected to possess the divisibility property; if an instance possesses this property, then a greedy algorithm gives an optimal solution. If $n \geqslant 40$, then the $O(n \log n)$ algorithm solves all instances to optimality; for $n \geqslant 30$, dynamic programming applied to subset-sum suffices to solve the instances that are not solved by the $O(n \log n)$ algorithm; for $n \leqslant 20$, branch-and-bound is occasionally needed, but requires only a small number of nodes, and always less than 1 second of running time.

n	t	# $O(n \log n)$	# DP	maximum # nodes	# greedy optimal	# SA optimal	# LB tight
10	0.1	66	20	12	72	77	86
10	0.2	69	20	22	72	58	89
10	0.3	68	23	22	68	59	93
10	0.4	82	1	40	85	62	85
20	0.1	81	12	94	84	51	94
20	0.2	94	5	167	94	43	99
20	0.3	99	0	320	100	42	99
20	0.4	99	1	0	99	35	100
30	0.1	100	0	0	100	50	100
30	0.2	98	2	0	98	51	100
30	0.3	100	0	0	100	57	100
30	0.4	100	0	0	100	68	100
40	0.1	100	0	0	100	63	100
40	0.2	100	0	0	100	64	100
40	0.3	100	0	0	100	63	100
40	0.4	100	0	0	100	54	100
50	0.1	100	0	0	100	72	100
50	0.2	100	0	0	100	63	100
50	0.3	100	0	0	100	69	100
50	0.4	100	0	0	100	75	100
75	0.1	100	0	0	100	75	100
75	0.2	100	0	0	100	79	100
75	0.3	100	0	0	100	78	100
75	0.4	100	0	0	100	83	100
100	0.1	100	0	0	100	81	100
100	0.2	100	0	0	100	86	100
100	0.3	100	0	0	100	78	100
100	0.4	100	0	0	100	78	100
200	0.1	100	0	0	100	96	100
200	0.2	100	0	0	100	98	100
200	0.3	100	0	0	100	99	100
200	0.4	100	0	0	100	97	100
500	0.1	100	0	0	100	99	100
500	0.2	100	0	0	100	100	100
500	0.3	100	0	0	100	100	100
500	0.4	100	0	0	100	100	100
1000	0.1	100	0	0	100	100	100
1000	0.2	100	0	0	100	100	100
1000	0.3	100	0	0	100	100	100
1000	0.4	100	0	0	100	100	100

TABLE 5.1. Computational results.

6

Just-In-Time Scheduling

The just-in-time concept decrees not to accept ordered goods before their due dates in order to avoid inventory cost. This bounces the inventory cost back to the manufacturer: products completed before their due dates have to be stored, thereby entailing storage cost. Preclusion of early completion conflicts with the traditional policy of keeping work-in-process inventories down. This chapter addresses the single-machine scheduling problem of minimizing total inventory cost, comprising cost due to work-in-process inventories and storage cost due to early completions. The cost components are measured by the sum of the job completion times and by the sum of the job earlinesses. This problem is known to be $\mathcal{NP}$-hard. In Section 6.1, we describe the problem in detail and give an overview of the literature on scheduling problems with earliness penalties.

The problem differs from traditional scheduling problems, as machine idle time between the execution of jobs may reduce total inventory cost. The search for an optimal schedule can still be limited to the set of job sequences; for any given sequence, a polynomial-time algorithm can be applied to insert machine idle time between the jobs so as to minimize total inventory cost. This algorithm is presented in Section 6.2.

The allowance of machine idle time between the execution of jobs singles out our problem from most concurrent research on problems with earliness penalties. To our knowledge, we present the first branch-and-bound algorithm for a single-machine scheduling problem where machine idle time between the jobs is allowed. In Section 6.3, we discuss the components of the branch-and-bound algorithm including the upper bound, the branching rule, the search strategy, and the many dominance rules. The derivation of lower bounds is significantly complicated by the possibility of machine idle time. The range of the due dates in proportion to the processing times mainly determines how much idle time is desired; this gives rise to many different classes of problem instances. To cope

with the different problem instances, we present five approaches for lower bound computation; each of these is only suitable for a specific class of problem instances. Lagrangian relaxation is one of them; its application proceeds in the spirit of Chapter 2. The lower bounds are presented in Section 6.4. The branch-and-bound algorithm is inelegant: it is based upon many dominance rules and various lower bound approaches. Unfortunately, it is not very efficient either; the computational results presented in Section 6.5 exhibit that we can solve only specific instances with up to 20 jobs. Conclusions are given in Section 6.6.

6.1. Introduction

The just-in-time concept has affected the attitude towards inventories significantly. In order to keep inventories down, there is a reluctance to accept ordered goods prior to their due dates. This implies that manufacturers have to store early goods before they can be shipped to their destinations. This has added a relatively new aspect to machine scheduling theory: the preclusion of earliness. In principle, earliness can be avoided by allowing machine idle time, thereby delaying jobs. Machine idleness, however, runs counter to minimizing work-in-process inventories, to maximizing machine utilization, and to observing due dates.

Within this context, we address the following situation. A set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n independent jobs has to be scheduled on a single machine, which is continuously available from time 0 onwards. The machine can handle at most one job at a time. Job J_j ($j = 1, \ldots, n$) requires processing during an uninterrupted period of length p_j and is ideally completed exactly on its due date d_j. We may assume that the processing times and the due dates are integral. A *schedule* specifies for each job J_j a completion time C_j such that the jobs do not overlap in their execution. The order in which the machine processes the jobs is called the *job sequence.* For a given schedule, the earliness of J_j is defined as $E_j = \max\{d_j - C_j, 0\}$ and its tardiness as $T_j = \max\{C_j - d_j, 0\}$. In addition, we define *maximum earliness* as $E_{\max} = \max_{1 \leqslant j \leqslant n} E_j$ and *maximum tardiness* as $T_{\max} = \max_{1 \leqslant j \leqslant n} T_j$. Accordingly, J_j is called *early*, *just-in-time*, or *tardy*, if $C_j < d_j$, $C_j = d_j$, or $C_j > d_j$, respectively.

Since earliness is a performance measure that is non-increasing in the job completion times, permitting machine idle time is advantageous. The inclusion of the acronym *nmit* in the second field of the three-field notation introduced in Section 1.1.4 signifies that no machine idle time is allowed.

Three single-machine scheduling problems involving job earliness have been considered in the literature. The best-known is the minimization of $E_{\max}$. If machine idle time is not allowed, then the problem is solved by scheduling the jobs in non-decreasing order of $d_j - p_j$; this is known as the *minimum slack time order.* If machine idle time is allowed, then the problem is trivial: for any given sequence, we delay the jobs until all are just-in-time or tardy. This approach also applies to $1||\Sigma E_j$, but, surprisingly, $1|nmit|\Sigma E_j$ is $\mathcal{NP}$-hard in the ordinary sense [Du and Leung, 1990]. The third problem is to maximize $\Sigma_{j=1}^{n} w_j E_j$, where w_j is the weight of J_j, denoted as $1||-\Sigma w_j E_j$; it is solvable in pseudo-polynomial time by an algorithm due to Lawler and Moore [1969]. Note that the

objective function is non-decreasing in the job completion times.

The combination of earliness with another performance measure, reflecting other considerations, takes us into the arena of bicriteria scheduling. The state of the art, as far as a measure of earliness is concerned, is as follows. For the $1|pmtn, nmit|\alpha\Sigma C_j+\beta E_{\max}$ problem, Hoogeveen and Van de Velde [1990] present an algorithm that runs in $O(n^4)$ time. They show that the same algorithm also solves $1||\alpha\Sigma C_j+\beta E_{\max}$ in the case that $\alpha \geqslant \beta$. Hoogeveen [1990] presents an algorithm that solves $1||\alpha E_{\max}+\beta T_{\max}$ and $1|nmit|F(E_{\max}, T_{\max})$ in $O(n^2 \log n)$ time; F is here an arbitrary non-decreasing function of $E_{\max}$ and $T_{\max}$. For the $1|nmit|\Sigma(\alpha_j E_j+\beta_j T_j)$ problem, Ow and Morton [1989] propose a local search method to generate approximate solutions. A voluminous part of research is concerned with common due date scheduling. Here, we have $d_j = d$ for each J_j ($j = 1, \ldots, n$); the objective is to minimize some function of earliness and tardiness. A survey of problems, algorithms, and computational complexity is provided by Baker and Scudder [1990].

In this chapter, we consider the problem of minimizing total inventory cost, comprising two components: cost due to work-in-process inventory and storage cost due to early completions. These components are assumed to depend linearly on the sum of the job completion times and the sum of the job earlinesses. If α and β denote the cost per job per unit time for work-in-process inventory and storage of finished product, respectively, then the total inventory cost for a given schedule σ is

$$f(\sigma) = \alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} E_j.$$

Without loss of generality, we assume α and β to be integral, positive, and relatively prime. Since we have by definition that $E_j = T_j - C_j + d_j$ for $j = 1, \ldots, n$, the objective function can alternatively be written as

$$(\alpha-\beta)\sum_{j=1}^{n} C_j + \beta\sum_{j=1}^{n} (T_j + d_j).$$

If $\alpha > \beta$, then this objective function is increasing in the job completion times; hence, any optimal schedule has no machine idle time. The case $\alpha = \beta$ reduces to $1||\Sigma T_j$, which is $\mathcal{NP}$-hard in the ordinary sense [Du and Leung, 1990]. Garey, Tarjan, and Wilfong [1988] prove that the case $\alpha < \beta$ is $\mathcal{NP}$-hard too. We note that the case $n\alpha < \beta$ reduces to $1|r_j|\Sigma C_j$, which is $\mathcal{NP}$-hard in the strong sense [Lenstra, Rinnooy Kan, and Brucker, 1977].

In this chapter, we examine the case $\alpha < \beta$ with machine idle time allowed. Each J_j ($j = 1, \ldots, n$) is then ideally completed exactly on its due date d_j. The purpose is to find a feasible schedule σ that minimizes $f(\sigma)$. Fry and Keong Leong [1987A], formulating the problem as an integer linear program, use a standard code for integer linear programming to find an optimal solution. Not surprisingly, they report that their approach is effective for instances with up to 12 jobs only.

6.2. The insertion of idle time for a given sequence

The search for an optimal schedule can be reduced to a search over the $n!$ different job sequences, as there is a clear-cut procedure to insert machine idle time so as to minimize total cost for a *given* sequence.

This procedure, however, is not new. Similar methods have been presented (cf. Baker and Scudder [1990]), including the ones proposed by Fry and Keong Leong [1987B] for the $1||\Sigma(\alpha C_j+\beta E_j+\gamma T_j)$ problem and by Garey, Tarjan, and Wilfong [1988] for the $1||\Sigma(E_j+T_j)$ problem. This is not surprising: as we have already noted, $T_j = C_j + E_j - d_j$ for all j; for specific choices for α and β, our problem is equivalent with theirs.

Suppose that the scheduling order is $\sigma=(J_n,\ldots,J_1)$. Accordingly, $\overline{C}_j = \Sigma_{k=j}^{n} p_k$ is the earliest possible completion time of J_j in this sequence. We introduce a vector $x=(x_1,\ldots,x_n)$ of variables, with x_j ($j=1,\ldots,n$) denoting the amount of idle time immediately before the execution of J_j. The actual completion time of J_j is then $C_j=\overline{C}_j+\Sigma_{k=j}^{n}x_k$. The problem of minimizing inventory cost for the given job sequence is then equivalent to determining values x_j ($j=1,\ldots,n$) that minimize

$$\alpha\sum_{j=1}^{n}(\overline{C}_j+\sum_{k=j}^{n}x_k)+\beta\sum_{j=1}^{n}\max(0,d_j-\overline{C}_j-\sum_{k=j}^{n}x_k)$$

subject to

$$x_j \geqslant 0, \qquad \text{for } j=1,\ldots,n.$$

By the introduction of auxiliary variables E_j denoting the earliness of J_j ($j=1,\ldots,n$), we can easily transform this problem into a linear programming problem. We know therefore that the optimum is attained in a vertex of the unspecified LP polytope. In addition, we know that the optimal x_j are integral, since the due dates, the processing times, α, and β are integral. A necessary condition for x to be optimal is that all existing primitive directional derivatives at x are non-negative (cf. Section 1.3.3). For this particular problem, the primitive directional derivatives are equal to the change of the scheduling cost if x_j is increased by one unit, and the change of the scheduling cost if x_j is decreased by one unit, for $j=1,\ldots,n$. The increase of x_j by one unit only affects J_j and the jobs succeeding J_j up to the first period of machine idle time after J_j. We call these jobs the *immediate successors* of J_j. Let Q_j denote the set containing J_j and its immediate successors, let n_j be the number of early jobs in Q_j, and let g_j be the primitive directional derivative for increasing x_j. We have then that $g_j=\alpha\,|Q_j|-\beta n_j$. Recall that each J_j is ideally completed on its due date d_j.

Using the above observations, we develop an inductive procedure for finding an optimal schedule for σ. This procedure finds an optimal schedule for the subsequence $(J_l,\ldots,J_1)$, given an optimal schedule for the subsequence $(J_{l-1},\ldots,J_1)$, for $l=2,\ldots,n$. The first step is to find out whether putting $C_l=d_l$ is feasible; if it is, then we have an optimal schedule for $(J_l,\ldots,J_1)$. Suppose $C_l=d_l$ is not feasible, because J_l overlaps with some other job. We then tentatively put $C_l=C_{l-1}-p_{l-1}$, and compute the optimal delay of the jobs in Q_l,

disregarding the jobs not in Q_l. The optimal delay, denoted by δ, is specified by the first point where g_l becomes non-negative. This delay is feasible if δ is not larger than the length of the period of idle time immediately after the last job in Q_l; let this length be $\delta_{\max}$. If $\delta \leqslant \delta_{\max}$, then we get an optimal schedule for $(J_l, \ldots, J_1)$ by delaying the jobs in Q_l by δ. If $\delta > \delta_{\max}$, then we delay the jobs in Q_l by $\delta_{\max}$. At this point, we repeat the process for J_l: we update Q_l, and evaluate if further delay of the jobs in Q_l is advantageous. We now give a step-wise description of the idle time insertion algorithm.

IDLE TIME INSERTION ALGORITHM

Step 0. $C_1 = d_1$; $l = 2$.

Step 1. If $l = n + 1$, go to Step 9.

Step 2. Put $C_l = \min\{d_l, C_{l-1} - p_{l-1}\}$. If $C_l = d_l$, then go to Step 8.

Step 3. Determine Q_l and evaluate g_l. If $g_l \geqslant 0$, then go to Step 8.

Step 4. Compute E_j for each job $J_j \in Q_l$.

Step 5. Compute $\delta_{\max}$, i.e., the length of the period of idle time immediately after the last job in Q_l.

Step 6. Compute $a = \lfloor (|Q_l|)\alpha/\beta \rfloor$. Let $k = |Q_l| - a$. Determine the kth smallest value of the earlinesses of the jobs in Q_l; this value is denoted as $E_{[k]}$. If the jobs in Q_l are delayed by $\delta = E_{[k]}$, then at most a jobs in Q_l remain early; due to the choice of a, g_l then becomes non-negative.

Step 7. Delay the jobs in Q_l by $\Delta = \min\{\delta, \delta_{\max}\}$. If $\delta > \delta_{\max}$, then go to Step 3.

Step 8. $l \leftarrow l + 1$; go to Step 1.

Step 9. An optimal schedule for the sequence $(J_n, \ldots, J_1)$ has been determined.

THEOREM 6.1. *The idle time insertion algorithm generates an optimal schedule for a given sequence.*

PROOF. The proof proceeds by induction. The algorithm clearly produces the optimal schedule in case of a single job. Suppose now we want to find an optimal schedule for the sequence $(J_l, \ldots, J_1)$, having an optimal schedule for the sequence $(J_{l-1}, \ldots, J_1)$ available. There are two cases to consider. First, suppose $d_l \leqslant C_{l-1} - p_{l-1}$; in this case, we let $C_l = d_l$, and retain the completion times of the other jobs; this specifies an optimal schedule for the sequence $(J_l, \ldots, J_1)$. Suppose now $d_l > C_l - p_l$; for this case, delaying J_{l-1} and thereby its immediate successors, i.e., the jobs contained in the set Q_{l-1}, may be advantageous. We can compute the cost of delaying Q_{l-1} by one unit; we know that the benefit of delaying J_l by one unit is equal to $\beta - \alpha$. If the cost is higher than or equal to the benefit, then we put $C_l = C_{l-1} - p_{l-1}$, and we have an optimal schedule for $(J_l, \ldots, J_1)$; otherwise, we postpone the jobs in Q_{l-1} by one unit, and evaluate whether further postponement is advantageous. The idle time insertion algorithm shortcuts this procedure by computing the break-even point, that is, the point where further delay is not advantageous. □

Consider the example for which the data are given in Table 6.1. Let $\alpha = 1$, and let $\beta = 4$. We construct the optimal schedule for the sequence (J_3, J_2, J_1). First, we put $C_1 = d_1 = 15$. Next, we let $C_2 = d_2 = 10$, as $d_2 \leqslant C_1 - p_1$. Note that $d_3 > C_2 - p_2$. Therefore, we tentatively put $C_3 = C_2 - p_2 = 7$, and consider delaying J_3 and J_2. Apparently, we have $Q_3 = \{J_3, J_2\}$, $n_3 = 1$, $g_3 = 2\alpha - \beta < 0$, and $E_{[2]} = 3$. However, $\delta_{\max} = C_1 - p_1 - C_2 = 2$, therefore, we delay J_2 and J_3 by 2 units. At this point, the three jobs are processed consecutively. Now we have $g_3 = 3\alpha - \beta$, and further delay is still advantageous. As $E_{[3]} = 1$, we insert one more unit of machine idle time. The optimal schedule for each subproblem is depicted in Figure 6.1.

J_j	p_j	d_j
J_1	3	15
J_2	3	10
J_3	6	10

TABLE 6.1. Data for the example.

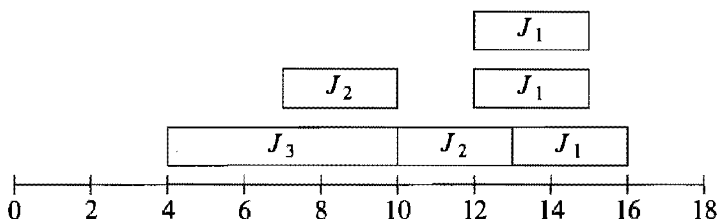


FIGURE 6.1. Schedules for the example.

The algorithm runs in $O(n^2)$ time. A complete run through the main part of the algorithm, i.e., steps 2 through 8, takes $O(n)$ time: this is needed to identify the set Q_l, to compute the primitive directional derivative g_l, the values $\delta_{\max}$ and δ, and to delay the jobs, if necessary. The value δ is determined in $O(n)$ time through a median-finding technique; see Aho, Hopcroft, and Ullman [1982]. After each run through the main part of the algorithm, a gap between two successive jobs is closed. As at most $n-2$ such gaps exist, the algorithm runs in $O(n^2)$ time. For the case $2\alpha = \beta$, i.e., for the problem $1 | | \Sigma(E_j + T_j)$, Garey, Tarjan, and Wilfong [1988] show that the idle time insertion procedure can be implemented to run in $O(n \log n)$ time.

The problem of inserting machine idle time is also solved by a symmetric procedure starting with the first job in σ. Because of our specific branching rule, however, we choose to start at the end.

In the remainder, we use the terms sequence and schedule interchangeably. Unless stated otherwise, σ also refers to the optimal schedule for the sequence σ and to the set of jobs in the sequence σ. Throughout the chapter, we let $p(\sigma) = \Sigma_{J_j \in \sigma} p_j$.

6.3. THE BRANCH-AND-BOUND ALGORITHM

We adopt a *backward sequencing branching rule*: a node at level k of the search tree corresponds to a sequence π with k jobs fixed in the last k positions. We assume from now on that the first job in a partial schedule π is not started before time $p(\mathcal{J}-\pi)$; this additional restriction, imposed to leave space for the remaining jobs, is easily incorporated in the idle time insertion algorithm. Let $f(\pi)$ denote the minimal inventory cost for π. Let $\bar{f}(\pi)$ denote the minimal inventory cost for π if the first job may start before time $p(\mathcal{J}-\pi)$; the notation $\bar{f}(\pi)$ is only needed in this section. For any partial schedule π, we have $f(\pi) \geqslant \bar{f}(\pi)$; equality holds for any complete schedule σ.

We employ a *depth-first* strategy to explore the tree: at each level, we generate the descendant nodes for only one node at a time. At level k, there are $n-k$ descendant nodes: one for each unscheduled job. The completion times for the jobs in π are only temporary. Branching from a node that corresponds to π, we add some job J_j leading to the sequence $J_j\pi$. Subsequently, we determine the associated optimal schedule for $J_j\pi$, and possibly delay some jobs in π. We branch from the nodes in order of non-increasing due dates of the associated jobs. Before entering the search tree, we determine an upper bound on the optimal solution value. We use the optimal schedule corresponding to the minimum slack time sequence as an initial solution, and try to reduce its cost by pairwise adjacent interchanges.

A node is discarded if its associated partial schedule π cannot lead to a complete schedule with cost less than UB; UB denotes the currently best solution value. Let $LB(\mathcal{J}-\pi)$ be some lower bound on the minimal cost of scheduling the jobs in the set $\mathcal{J}-\pi$. Obviously, we discard a node if $f(\pi)+LB(\mathcal{J}-\pi) \geqslant UB$. The following rule is usually overlooked. Let $g(\sigma_1,\sigma_2)$ be a lower bound on the cost for scheduling the jobs in σ_1 given the final partial schedule σ_2.

THEOREM 6.2. *The partial schedule π can be discarded if there exists a $J_j \in \mathcal{J}-\pi$ for which $\bar{f}(J_j\pi) + g(\mathcal{J}-\pi-J_j, \pi) \geqslant UB$.*

PROOF. Consider a complete sequence σ that has π as final subsequence. Thus, σ can be written as $\sigma = \pi_1 J_j \pi_2 \pi$. Accordingly, we have

$$f(\sigma) = f(\pi_1 J_j \pi_2 \pi) \geqslant \bar{f}(J_j\pi) + g(\pi_1\pi_2, \pi) \geqslant UB. \quad \square$$

It is essential that $g(\mathcal{J}-\pi-J_j,\pi)$ depends only on π and not on $J_j\pi$, and that we use $\bar{f}(J_j\pi)$ instead of $f(J_j\pi)$. We derive two corollaries from Theorem 6.2.

COROLLARY 6.1. *If for a given partial schedule π, we have that $\bar{f}(J_jJ_k\pi) + g(\mathcal{J}-\pi-J_j-J_k, \pi) \geqslant UB$ for some $J_j \in \mathcal{J}-\pi$ and $J_k \in \mathcal{J}$, then J_k precedes J_k in any complete schedule $\sigma\pi$ with $f(\sigma\pi) < UB$.* $\square$

COROLLARY 6.2. *The partial schedule π can be discarded if two jobs $J_j \in \mathcal{J}-\pi$ and $J_k \in \mathcal{J}-\pi$ exist with $g(\mathcal{J}-\pi-J_j-J_k,\pi) + \min\{\bar{f}(J_jJ_k\pi), \bar{f}(J_kJ_j\pi)\} \geqslant UB$.* $\square$

If a partial schedule $\pi^* \neq \pi$ exists comprising the same jobs as π and having $f(\sigma\pi^*) \leqslant f(\sigma\pi)$ for any sequence σ for the remaining $n-k$ jobs, then we can also discard π. If $f(\sigma\pi^*) < f(\sigma\pi)$ for some σ, then π is *dominated* by π^*. If $f(\sigma\pi^*) = f(\sigma\pi)$ for every σ, then we discard either π^* or π. The dominance condition above can be narrowed by the requirement that $f(\pi^*) \leqslant f(\pi)$ and that the circumstances to add the remaining $n-k$ jobs to π^* are at least as good as the circumstances to add the remaining jobs to π. The question whether such a sequence π^* exists is of course already $\mathcal{NP}$-complete. We strive therefore to identify sufficient conditions to discard π. The temporary nature of the job completion times for π complicates the achievement of this goal. We have to be careful with dominance conditions that are based on interchange arguments: the conditions must remain valid if the jobs in π are delayed.

Suppose the jobs in π have been reindexed in order of increasing completion times. In each of the following theorems, stating the dominance rules, the sequence π^* is obtained from π by swapping two jobs, say, J_j and J_k. We do not compute the optimal completion times for the sequence π^*. Instead, we determine the job completion times for the sequence π^* as follows. Let C_i and C_i^* be the completion time of J_i in the schedule π and π^*, respectively. Then we let

$$\begin{aligned} C_i^* &= C_i, && \text{for } i = 1, \ldots, j-1,\ i = k+1, \ldots, |\pi|, \\ C_i^* &= C_i - p_j + p_k, && \text{for } i = j+1, \ldots, k-1, \\ C_k^* &= C_j - p_j + p_k, \\ C_j^* &= C_k. \end{aligned}$$

Let $F(\pi^*)$ be the cost associated with the completion times C_i^*, for $i = 1, \ldots, |\pi|$. Hence, $F(\pi^*) \geqslant f(\pi^*)$. To validate the following dominance rules, we must verify that $f(\pi) \geqslant F(\pi^*)$, even if the jobs are delayed. Due to the relation between π and π^*, this comes down to verifying that for each $\Delta \geqslant 0$

$$\alpha \sum_{i=j}^{k} C_i + \beta \sum_{i=j}^{k} \max\{0, d_i - C_i - \Delta\} \geqslant \alpha \sum_{i=j}^{k} C_i^* + \beta \sum_{i=j}^{k} \max\{0, d_i - C_i^* - \Delta\}. \qquad (6.1)$$

We start with a straightforward result.

THEOREM 6.3. *There is an optimal schedule with J_j preceding J_k if $p_j = p_k$ and $d_j \leqslant d_k$.* □

THEOREM 6.4. *The partial sequence π can be discarded if there are two jobs J_j and J_k with $C_k = C_j + \Sigma_{i=j+1}^{k} p_i$ for which*

$p_j > p_k$, *and*

$$\alpha \sum_{i=j}^{k} C_i + \beta \sum_{i=j+1}^{k} \max\{0, d_i - C_i\} \geqslant \alpha \sum_{i=j}^{k} C_i^* + \beta \sum_{i=j+1}^{k} \max\{0, d_i - C_i^*\}. \qquad (6.2)$$

PROOF. Define $c(\Delta)$ as the change of cost due to the interchange, after delaying the jobs by $\Delta \geqslant 0$; i.e.,

$$c(\Delta) = \alpha \sum_{i=j}^{k} C_i + \beta \sum_{i=j}^{k} \max\{0, d_i - C_i - \Delta\} - \alpha \sum_{i=j}^{k} C_i^* - \beta \sum_{i=j}^{k} \max\{0, d_i - C_i^* - \Delta\}.$$

We prove that $c(\Delta) \geqslant 0$ for all $\Delta \geqslant 0$. From condition (6.2), it follows immediately that $c(0) \geqslant 0$. Furthermore, $C_j < C_j^*$ implies $\max\{0, d_j - C_j - \Delta\} \geqslant \max\{0, d_j - C_j^* - \Delta\}$ for all $\Delta \geqslant 0$; $C_i > C_i^*$ for $i = j+1, \ldots, k$ implies $\max\{0, d_i - C_i - \Delta\} - \max\{0, d_i - C_i^* - \Delta\} \geqslant \max\{0, d_i - C_i\} - \max\{0, d_i - C_i^*\}$ for all $\Delta \geqslant 0$. Combining the inequalities, we get the desired result. □

The possible increase of E_j is excluded here. The following theorem shows that in case no idle time exists between two adjacent jobs, then dominance already exists if condition (6.1) is satisfied for $\Delta = 0$.

THEOREM 6.5. *The partial sequence π can be discarded if there are two jobs J_j and J_k with $C_k = C_j + p_k$ for which*

$$p_j > p_k,$$

and

$$\alpha(p_j - p_k) + \beta \max\{0, d_j - C_j\} + \beta \max\{0, d_k - C_k\} \geqslant$$
$$\beta \max\{0, d_j - C_k\} + \beta \max\{0, d_k - C_k + p_j\}. \tag{6.3}$$

PROOF. Define $c(\Delta)$ as the change of cost due to the interchange, after delaying the jobs by $\Delta \geqslant 0$; i.e.,

$$c(\Delta) = \alpha(p_j - p_k) + \beta \max\{0, d_j - C_j - \Delta\} - \beta \max\{0, d_j - C_k - \Delta\} +$$
$$\beta \max\{0, d_k - C_k - \Delta\} - \beta \max\{0, d_k - C_k + p_j - \Delta\}.$$

We need to show that the condition (6.3), stating that $c(0) > 0$, implies $c(\Delta) \geqslant 0$ for all $\Delta \geqslant 0$. Note that $\alpha < \beta$ implies that at least one due date is smaller than C_k; otherwise, condition (6.3) is not valid.

The expression $c(\Delta)$ has three components. The first component is $\alpha(p_j - p_k)$; it is a constant. The second component is $\beta \max\{0, d_j - C_j - \Delta\} - \beta \max\{0, d_j - C_k - \Delta\}$; it is a piecewise linear function of Δ. The function value is βp_k if $d_j \geqslant C_k + \Delta$; if $C_k + \Delta > d_j \geqslant C_j + \Delta$, then the gradient is -1. The function value is 0 if $d_j \leqslant C_j + \Delta$. The third component is $\beta \max\{0, d_k - C_k - \Delta\} - \beta \max\{0, d_k - C_k + p_j - \Delta\}$; it is also a piecewise linear function of Δ. The function value is $-\beta p_j$ if $d_k \geqslant C_k + \Delta$. The gradient is 1 if $C_k + \Delta > d_k \geqslant C_k - p_j + \Delta$. The function value is 0 for $d_k \leqslant C_k - p_j + \Delta$. Combining the three components yields a piecewise linear function whose behavior depends on the due dates. We now make the following observations. First, $c(\Delta) > 0$ if $\Delta \geqslant d_k - C_k + p_j$. Second, if $c(t) > 0$ for some $t \geqslant d_k - C_k$, then $c(\Delta) > 0$ for all $\Delta \geqslant t$. As at least one due date is smaller than C_k, the second observation implies that, if $d_k \leqslant d_j$, then $c(\Delta) > 0$ for all $\Delta \geqslant 0$.

The only case left to consider is $d_j < d_k$ and $0 \leqslant \Delta \leqslant d_k - C_k$. Then, we have $c(\Delta) = \alpha(p_j - p_k) - \beta p_j + \beta \max\{0, d_j - C_j - \Delta\}$. As $d_j - C_j - \Delta \leqslant d_j - C_j = d_j - C_k + p_k \leqslant p_k$, we get $c(0) \leqslant (\alpha - \beta)(p_j - p_k) \leqslant 0$, which contradicts the assumption. This completes the proof. □

In Corollary 6.3, explicit conditions for the existence of dominance are derived from Theorem 6.5. This corollary is referred to when lower bounds are discussed in Section 6.4.

COROLLARY 6.3. *The partial sequence π can be discarded if there are two jobs J_j and J_k with $C_k = C_j + p_k$ such that*

$$p_j > p_k,$$

and one of the following conditions is satisfied:

$$C_k - p_j \geqslant d_k,$$
$$C_k - p_j < d_k,\ C_k \geqslant d_k,\ C_j \geqslant d_j, \text{ and } \alpha(p_j - p_k) \geqslant \beta(d_k - C_k + p_j),$$
$$C_k - p_j < d_k,\ C_k < d_k,\ C_j \geqslant d_j, \text{ and } \alpha(p_j - p_k) \geqslant \beta p_j,$$
$$C_k - p_j < d_k,\ C_k \geqslant d_k,\ C_j < d_j, \text{ and } \alpha(p_j - p_k) \geqslant \beta(d_k - d_j - p_k + p_j).$$

□

THEOREM 6.6. *The partial sequence π with J_k scheduled last is dominated if there is a J_j such that*

$$p_j > p_k, \text{ and } C_j - p_j + p_k \geqslant d_k.$$

PROOF. Let $\pi = \pi_1 J_j \pi_2 J_k$ and $\pi^* = \pi_1 J_k \pi_2 J_j$. We compute the effect of the interchange on the scheduling cost. Since J_k is the last job in the optimal schedule π, we have $C_k \geqslant d_k$. In addition, we know $C_j{}^* = \max\{d_j, C_k - p_k + p_j\}$ and $C_k{}^* = C_j - p_j + p_k \geqslant d_k$. First, suppose $C_j{}^* = d_j$. The effect of the interchange is then equal to

$$\alpha(C_j + C_k - (C_j - p_j + p_k) - d_j) + \beta(d_j - C_j) \geqslant$$
$$\alpha(C_k + p_j - p_k - d_j) + \alpha(d_j - C_j) > 0,$$

as $C_k - p_k \geqslant C_j$. Second, suppose that $C_j{}^* = C_k - p_k + p_j$. The effect of the interchange is then equal to

$$\alpha[C_j + C_k - (C_k - p_k + p_j) - (C_j - p_j + p_k)] + \beta \max\{0, d_j - C_j\} \geqslant 0.$$

The effect remains non-negative if the jobs are delayed. □

THEOREM 6.7. *There is an optimal schedule in which J_k is not scheduled in the last position, if there is some J_j with $p_j > p_k$ and $d_j - p_j \geqslant d_k - p_k$.*

PROOF. We let $\pi = \pi_1 J_j \pi_2 J_k$ and $\pi^* = \pi_1 J_k \pi_2 J_j$ and compute the effect of the interchange. We have $C_k \geqslant d_k$ and $C_k - p_k \geqslant C_j$; in addition, we define here $C_j{}^* = \max\{d_j, C_k - p_k + p_j\}$. The effect of the interchange has to be non-negative; we therefore have to prove that

$$\alpha C_k + \beta \max\{0, d_j - C_j\} \geqslant \alpha(p_k - p_j + C_j{}^*) + \beta \max\{0, d_k - p_k + p_j - C_j\}. \quad (6.4)$$

First, we examine the case $C_j^* = C_k - p_k + p_j$. Expression (6.4) is then equivalent to

$$\beta \max\{0, d_j - C_j\} \geq \beta \max\{0, d_k - p_k + p_j - C_j\},$$

which is true for any C_j since $d_j - p_j \geq d_k - p_k$. Second, consider the case $C_j^* = d_j$. This implies $d_j > C_j$, since $d_j \geq C_k - p_k + p_j > C_j - p_k + p_j > C_j$. Hence, expression (6.4) is equivalent to

$$\alpha C_k + \beta(d_j - C_j) \geq \alpha(p_k - p_j + d_j) + \beta \max\{0, d_k - p_k + p_j - C_j\}.$$

Suppose $\max\{0, d_k - p_k + p_j - C_j\} = d_k - p_k + p_j - C_j$. We must then verify that

$$\alpha C_k + \beta d_j \geq \alpha(d_j - p_j + p_k) + \beta(d_k - p_k + p_j).$$

As $C_k \geq d_k$, we only need to prove that

$$0 \geq (\alpha - \beta)[(d_j - p_j) - (d_k - p_k)];$$

this expression is true since $\beta > \alpha$ and $d_j - p_j \geq d_k - p_k$. Conversely, suppose $\max\{0, d_k - p_k + p_j - C_j\} = 0$. Since $\alpha C_k + \beta(d_j - C_j) \geq \alpha(C_k + d_j - C_j) \geq \alpha(p_k + d_j) > \alpha(p_k - p_j + d_j)$, expression (6.4) is also true for this case. □

COROLLARY 6.4. *There is an optimal schedule in which J_j is scheduled last if $p_j \geq p_k$ and $d_j - p_j \geq d_k - p_k$ for each $J_k \in \mathcal{J}$.* □

6.4. LOWER BOUNDS

In this section, we present five lower bound procedures. It seems to be impossible to develop a lower bound procedure that copes satisfactorily with all conceivable due date patterns. For example, imagine an instance with due dates small with respect to the sum of the processing times; little idle time needs then to be inserted. In contrast, consider an instance with $d_k \gg \Sigma_{j=1}^n p_j$ for each J_k; the machine will then be idle for some time before processing the first job. Numerous variations and combinations of both patterns are possible.

Each of the lower bound methods is effective for a specific class of instances. Nonetheless, we use them supplementary rather than complementary. We partition the job set $\mathcal{J}$ into subsets, apply each lower bound method to each subset, and aggregate the best lower bounds. In this way, we hope to obtain a stronger lower bound than the lower bounds obtained for the entire set $\mathcal{J}$. Success depends on the partitioning strategy. The jobs in a subset should be conflicting. If they are not, then all jobs are completed exactly on their due dates, giving rise to the weak lower bound $\alpha \Sigma_{j=1}^n d_j$. In this sense, we prefer subsets such that the executions of the jobs in the same subset interfere with each other, but not with the execution of the jobs in the other subsets. We propose two partitioning strategies that pursue this effect.

The first strategy is motivated by the structure of an optimal schedule. The jobs that are consecutively processed between two periods of idle time interfere with each other, but not with the other jobs. Such a partitioning is hard to obtain. To mimic such a partitioning, we identify clusters. A *cluster* is a set of jobs such that for each job J_j in the cluster there is another job J_k in the cluster

such that the intervals $[d_j-p_j, d_j]$ and $[d_k-p_k, d_k]$ overlap; hence, for each job in the cluster there exists a conflict with at least one other job in the cluster. However, clusters may interfere with each other in any optimal schedule.

The second strategy is the following. Given a partial schedule π, we try to identify the jobs not in π that will be early in any optimal complete schedule of the form $\sigma\pi$. We call these jobs *surely* early. The idea is to derive an upper bound T on the completion times of the unscheduled jobs; accordingly, $J_j \in \mathcal{J}-\pi$ is surely early if $d_j > T$. For instance, let g be the primitive directional derivative for delaying the first job in π by one unit. Suppose $|\mathcal{J}-\pi|(\beta-\alpha) \leqslant g$. The current set of completion times for the jobs in π is then optimal for any schedule $\sigma\pi$; an upper bound T is then the start time of the first job in π. Other upper bounds are derived from the dominance rules. Suppose J_j and J_k are adjacent in π with $p_j > p_k$ and J_j preceding J_k. (It is not necessary that $C_k = C_j + p_k$.) The first condition of Corollary 6.3 indicates that π is dominated if $C_k \geqslant d_k + p_j$; hence, an upper bound is given by $d_k + p_j - 1 - \Sigma_{J_i \in \pi, C_i \leqslant C_k} p_i$. From the other criteria in Corollary 6.3 and from Theorem 6.7, similar upper bounds are derived. They can also be derived from Theorem 6.4, but this requires an intricate procedure. Finally, we set T equal to the minimum of all upper bounds. If no upper bound is specified, then we let $T = \infty$.

6.4.1. First method: relax the objective function

Let $\mathcal{E}$ denote the set of surely early jobs; let $\mathcal{R}$ be the set of remaining jobs. Observe that

$$\min_{\sigma \in \Omega} f(\sigma) \geqslant \min_{\sigma \in \Omega_{\mathcal{R}}} \sum_{J_j \in \mathcal{R}} \alpha C_j + \min_{\sigma \in \Omega_{\mathcal{E}}} \sum_{J_j \in \mathcal{E}} [\alpha C_j + \beta E_j],$$

where $\Omega_{\mathcal{R}}$ and $\Omega_{\mathcal{E}}$ denote the set of feasible schedules for the jobs in $\mathcal{R}$ and $\mathcal{E}$. The problem of minimizing $\Sigma_{J_j \in \mathcal{E}} [\alpha C_j + \beta E_j]$ is solvable in polynomial time. Since we have $E_j = d_j - C_j$ for each $J_j \in \mathcal{E}$, the scheduling cost reduces to $\Sigma_{J_j \in \mathcal{E}} [(\alpha-\beta)C_j + \beta d_j]$. Applying an analogon of Smith's rule [Smith, 1956], we minimize this cost component by scheduling the jobs in $\mathcal{E}$ in the interval $[T-p(\mathcal{E}), T]$ in order of non-increasing processing times; the correctness of this rule is easily verified by an interchange argument. The other subproblem is solved by Smith's rule: simply schedule the jobs in $\mathcal{R}$ in non-decreasing order of their processing times in the interval $[0, p(\mathcal{R})]$. In the example, $\mathcal{E} = \varnothing$, and the lower bound is 21α.

A slight improvement of the lower bound is possible. Let $E_{\max}{}^*$ be the minimum maximum earliness for the jobs in $\mathcal{R}$ if they are processed in the interval $[0, p(\mathcal{R})]$. We can compute $E_{\max}{}^*$ from the minimum-slack-time sequence, that is, the sequence in which the jobs appear in order of non-decreasing values $d_j - p_j$. Avoiding $E_{\max}{}^*$ requires at least $E_{\max}{}^*$ units of machine idle time. The lower bound can therefore be improved by $\alpha E_{\max}{}^*$. If we have stored the shortest-processing-time sequence and the minimum-slack-time sequence, then we compute this lower bound in $O(n)$ time per node. In the example, we have $E_{\max}{}^* = 4$; hence, the lower bound is 25α. This lower bound approach can only be applied in conjunction with Theorem 6.2 if $\mathcal{E} = \varnothing$.

Since all jobs in $\mathfrak{R}$ are scheduled in the interval $[0,p(\mathfrak{R})]$, and since only one early job in $\mathfrak{R}$ is taken into account, this lower bound is only effective if the due dates are small relative to the sum of the processing times.

6.4.2. Second method: relax the machine capacity

Recall that we write the objective function alternatively as $f(\sigma) = (\beta-\alpha)\Sigma_{j=1}^{n} E_j + \alpha\Sigma_{j=1}^{n} T_j + \alpha\Sigma_{j=1}^{n} d_j$ for each $\sigma \in \Omega$. Since the job earlinesses and tardinesses are non-negative by definition, we have that $f(\sigma) \geqslant \alpha\Sigma_{j=1}^{n} d_j$ for each $\sigma \in \Omega$.

We gain more insight deriving this bound in the following way. Suppose the machine can process an infinite number of jobs at the same time; this is a relaxation of the limited capacity of the machine. As $\alpha < \beta$, the optimal schedule has $C_j = d_j$ for each J_j; this gives rise to the lower bound $\alpha\Sigma_{j=1}^{n} d_j$. If none of the jobs overlap in their execution, then this schedule is feasible and hence optimal for the original problem. For the example, this relaxation gives the lower bound 35α. The corresponding schedule is not feasible: J_2 and J_3 overlap in their execution (see Figure 6.2).

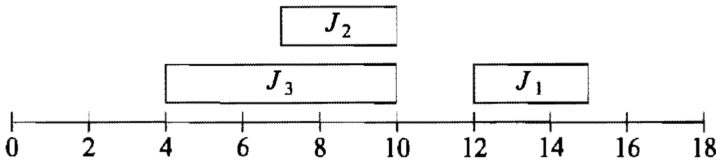


FIGURE 6.2. Gantt chart for machine with infinite capacity.

This conflict can be settled by executing J_3 before J_2, or, conversely, J_2 before J_3. If we intend to schedule J_2 after J_3, then we have basically two options: we retain either the completion time of J_3 or the completion time of J_2. For the first option, the additional cost is 3α; for the second option, the additional cost is $3(\beta-\alpha)$. Executing J_2 after J_3 costs therefore at least 3γ extra, where $\gamma = \min\{\alpha, \beta-\alpha\}$. Similarly, we find that executing J_3 after J_2 costs 6γ extra. Hence, the *minimum* additional cost required to settle the overlap is $\min\{3\gamma, 6\gamma\} = 3\gamma$. Accordingly, an improved lower bound is 38α.

We now describe a general procedure to improve the lower bound $\alpha\Sigma_{j=1}^{n} d_j$ by taking the overlap between jobs into consideration. Overlap of J_j and J_k ($J_j \neq J_k$) occurs if the intervals $[d_j-p_j,d_j]$ and $[d_k-p_k,d_k]$ overlap. Let $c_{jk} = \gamma\max\{0, d_j-(d_k-p_k)\}$ denote the *additional* cost to execute J_j immediately before J_k; let $\sigma(i) = j$ denote that J_j occupies the ith position in the sequence σ. For any optimal schedule σ, we have that $f(\sigma) \geqslant \alpha\Sigma_{j=1}^{n} d_j + \Sigma_{j=1}^{n-1} c_{\sigma(j)\sigma(j+1)}$; the last term is the length of the Hamiltonian path $\sigma(1) \cdots \sigma(n)$. The following procedure shows that the Hamiltonian path problem is solvable in $O(n\log n)$ time.

Partition the set of jobs into a set of clusters $Q_1, \ldots, Q_m$ as described above. Let HP_l be the shortest Hamiltonian path for Q_l, and let $c(HP_l)$ denote its length. We have $c(HP_l) = \gamma(p(Q_l) - \max_{J_j, J_k \in Q_l, J_k \neq J_j} c_{jk})$, for each l

($l = 1, \ldots, m$). We have also $\Sigma_{j=1}^{n-1} c_{\pi(j)\pi(j+1)} \geqslant \Sigma_{l=1}^{m} c(HP_l)$ for any sequence π, as can be easily verified. The individual Hamiltonian paths can be combined into one Hamiltonian path of length no more than the sum of the lengths of the separate paths.

6.4.3. Third method: relax the due dates

A major difficulty for the total inventory cost problem is that we cannot recognize a job as being early or tardy beforehand. However, if all jobs overlap, then we can. Relaxing the machine availability condition, we now assume that it is continuously available from time $d_{\min} - \Sigma_{j=1}^{n} p_j$ onwards, where $d_{\min} = \min_{1 \leqslant j \leqslant n} d_j$. Using the idle time insertion procedure, we know that there is an optimal schedule σ with some J_j completed exactly on its due date d_j. Due to the overlap, all jobs before J_j are early or just-in-time; all jobs after J_j are tardy or just-in-time. Reindexing the jobs in order of increasing completion times in σ, we see that

$$f(\sigma) = (\beta - \alpha) \sum_{i=1}^{j} [(i-1)p_i + d_i - d_j] + \alpha \sum_{i=j+1}^{n} [(n+1-i)p_i + d_j - d_i] + \alpha \sum_{i=1}^{n} d_i.$$

Rather than viewing $f(\sigma)$ as the sum of the job scheduling costs, we see it as the sum of the *positional* costs. The cost of assigning J_i ($i = 1, \ldots, n$) to the kth ($k = 1, \ldots, n$) position before d_j ($k = 1$ corresponds to the first position in σ) equals $(\beta - \alpha)[(k-1)p_i + d_i - d_j]$ and the cost of assigning J_i to the kth ($k = 1, \ldots, n-1$) position after d_j ($k = 1$ corresponds to the last position in σ) is equal to $\alpha[k p_i + d_j - d_i]$. The cost of assigning J_i to some position depends therefore only on p_i, on the position, and on J_j. For a given J_j, the problem of assigning jobs to positions is solved as a bipartite matching problem; this is done in $O(n^3)$ time by use of the Hungarian method (see Papadimitriou and Steiglitz [1982]). This leads to the following theorem.

THEOREM 6.8. *If all jobs overlap and if the machine is continuously available, then an optimal schedule for the problem of minimizing total inventory cost is obtained by taking the best solution after solving n assignment problems.* □

If $\bar{d}_j \leqslant d_j$ for each J_j ($j = 1, \ldots, n$), then we have for any schedule σ that

$$f(\sigma) = \alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} \max\{0, d_j - C_j\} \geqslant \alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} \max\{0, \bar{d}_j - C_j\}.$$

Hence we obtain a lower bound for the original problem by relaxing the due dates until all jobs overlap, and subsequently solving the problem under the assumption that the machine is continuously available. Such a procedure is time-consuming; we propose therefore two simplifications that are implemented to run in $O(n \log n)$ time per node. The first simplification concerns the case of equal due dates; the second one concerns the case of equal slack times. The two lower bound procedures are analyzed in Sections 6.4.3.1 and 6.4.3.2.

6.4.3.1. The common due date problem

Suppose the due dates have been replaced by a due date d common to all jobs. Consider the following *common due date problem*: for a given d, determine a schedule that minimizes

$$(\beta-\alpha)\sum_{j=1}^{n} E_j + \alpha\sum_{j=1}^{n} T_j + \alpha nd - \beta\sum_{j=1}^{n} \max\{0,d-d_j\}. \qquad \text{(CD)}$$

For any d, the optimal solution value is a lower bound for the original problem, since

$$\begin{aligned} f(\sigma) &= \alpha\sum_{j=1}^{n} C_j + \beta\sum_{j=1}^{n} \max\{0,d_j-C_j\} \\ &= \alpha\sum_{j=1}^{n} C_j + \beta\sum_{j=1}^{n} \max\{0,d-C_j-d+d_j\} \\ &\geqslant \alpha\sum_{j=1}^{n} C_j + \beta\sum_{j=1}^{n} \max\{0,d-C_j\} - \beta\sum_{j=1}^{n} \max\{0,d-d_j\} \\ &= (\beta-\alpha)\sum_{j=1}^{n} E_j + \alpha\sum_{j=1}^{n} T_j + \alpha nd - \beta\sum_{j=1}^{n} \max\{0,d-d_j\}. \end{aligned}$$

There are two issues involved: (i) how to solve problem (CD)?, and (ii) how to find the value d maximizing the lower bound?

Problem (CD) consists of two parts. The first part is the problem of minimizing $(\beta-\alpha)\Sigma_{j=1}^{n} E_j + \alpha\Sigma_{j=1}^{n} T_j$. If the machine is only available from time 0 onwards and if d is given, then this problem is $\mathcal{NP}$-hard [Hall, Kubiak, and Sethi, 1991; Hoogeveen and Van de Velde, 1991A]. However, a strong lower bound $L(d)$ is derived by applying Lagrangian relaxation (see Chapter 5). The second part is the easy problem of maximizing the function $G: d \rightarrow \alpha nd - \beta\Sigma_{j=1}^{n} \max\{0,d-d_j\}$. Rather than solving problem (CD) to optimality and finding the best d, we maximize the lower bound $L(d)+G(d)$ over d.

First, we derive the best Lagrangian lower bound $L(d)$ for a given d. The derivation proceeds without details; we refer to Chapter 5 for an elaborate treatment. Let $\mathcal{E}$ denote the set of early jobs. Since the machine is only available from time 0 onwards, we have the condition that $p(\mathcal{E}) \leqslant d$. We dualize this condition by use of the Lagrangian multiplier $\lambda \geqslant 0$. For a given $\lambda \geqslant 0$, the Lagrangian problem is then to find $L(d,\lambda)$, which is the minimum of

$$(\beta-\alpha)\Sigma_{j=1}^{n} E_j + \alpha\Sigma_{j=1}^{n} T_j + \lambda p(\mathcal{E}) - \lambda d.$$

The Lagrangian problem is solvable in polynomial time by Emmons' matching algorithm [Emmons, 1987], which proceeds by the concept of positional weights. Straightforward arguments show that there exists an optimal schedule with some job completed exactly on its due date. The weights for the early positions are then $\lambda, \lambda+(\beta-\alpha), \lambda+2(\beta-\alpha), \ldots, \lambda+(n-1)(\beta-\alpha)$; the smallest weight is for the first position in the schedule. The weights for the tardy positions are $\alpha, 2\alpha, \ldots, n\alpha$; the smallest weight is for the last position in the schedule. Emmons' matching algorithm assigns the job with the jth largest processing time

to the position with the jth smallest weight, for $j = 1, \dots, n$. Ties are settled to minimize the amount of work before d. Let σ_λ be the optimal schedule for the Lagrangian problem, and let $W(\sigma_\lambda)$ be the amount of work before d in σ_λ.

The best Lagrangian lower bound $L(d)$ is found as

$$L(d) = \max\{ L(d,\lambda) \mid \lambda \geqslant 0 \}.$$

Due to the integrality of α and β, the optimization over $\lambda \geqslant 0$ may be reduced to the optimization over $\lambda \in \mathbb{N}_0$. The optimal choice for λ can be shown to be such that $W(\sigma_{\lambda-1}) > d \geqslant W(\sigma_\lambda)$; this choice gives us the Lagrangian lower bound $L(d)$.

We are now able to characterize the function $L: d \to L(d)$. The function L is continuous and piecewise linear; the value $L(d)$ only depends on d through the choice for λ. Hence, there are at most $\min\{n^2, n\alpha\}$ breakpoints: they correspond to the values $d = W(\sigma_\lambda)$, for $\lambda = 0, 1, \dots, n\alpha$. The derivative of the trade-off curve between two consecutive breakpoints, the first corresponding to $W(\sigma_\lambda)$, is equal to $-\lambda$.

The function $G: d \to \alpha n d - \beta \Sigma_{j=1}^{n} \max\{0, d - d_j\}$ is also continuous and piecewise linear; the breakpoints correspond to the values $d = d_j$, for $j = 1, \dots, n$. The lower bound $L(d) + G(d)$ is therefore also continuous and piecewise linear in d; the value d maximizing this lower bound is found at a breakpoint.

For a given d, $L(d)$ is determined in $O(n \log n)$ time. The function L has $O(\min\{n^2, n\alpha\})$ breakpoints; the corresponding values are computed in $O(n^2)$ time. (Every new breakpoint is derived from the previous one by interchanging some jobs, requiring only constant time; $O(n^2)$ interchanges are needed to find all breakpoints.) The function G has $O(n)$ breakpoints. Hence, maximizing $L(d) + G(d)$ over d is achieved in $O(n^2)$ time.

We can also approximate the maximum of $L(d) + G(d)$ over d. A fair choice for d is the value that maximizes the function G. The maximum of G is attained at the kth smallest due date where $k = \lceil n\alpha / \beta \rceil$. For constant α, this lower bound is computed in $O(n \log n)$ time.

In our 3-job example, we have $d = 10$. For the positions after d, the weights are 1,2, and 3; for the positions before d, the weights are 0, 3, and 6. An optimal schedule is depicted in Figure 6.3. Its objective value is 39α; this happens to be the optimal solution value for the original problem.

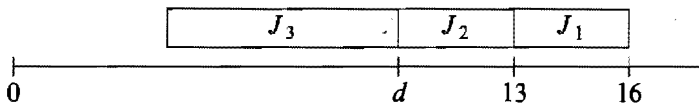


FIGURE 6.3. Optimal schedule for the common due date problem.

In a node of the search tree, there are two ways to implement this lower bound procedure. Let $\pi = \pi_1\pi_2$ be the partial schedule associated with the node. Disregarding π, we get the lower bound $f(\pi) + c(\mathcal{J} - \pi)$, where $c(\mathcal{J} - \pi)$ denotes the optimal solution value for the common due date problem for the jobs in $\mathcal{J} - \pi$.

However, if π_1 and the optimal schedule for the common due date problem overlap in their execution, then it makes sense to take π_1 into regard. We do this in the following way. First of all, we require that d is common to each $J_j \notin \pi_2$. Subsequently, we solve the common due date problem under the condition that the jobs in π_1 retain their positions. Given the set of positions, it is easy to construct an optimal schedule: assign the jobs in π_1 to the last $|\pi_1|$ positions, and assign the other jobs to the remaining positions according to Emmons' algorithm. Lemma 6.1 states that we may use the same set of positions as for the case $\pi_1 = \emptyset$.

LEMMA 6.1. *The optimal schedule for the common due date problem with the last* $|\pi_1|$ *jobs fixed occupies the* $\bar{n}$ *positions with least positional weights, where* $\bar{n} = n - |\pi_2|$.

PROOF. Suppose to the contrary that the optimal schedule σ for the jobs $J_j \notin \pi_2$ does not occupy the $\bar{n}$ positions with least positional weights. Let n_1 jobs in σ be early or just-in-time and let $n_2 = \bar{n} - n_1$ jobs in σ be tardy. Suppose the set of optimal weights corresponds to $\bar{n}_1$ positions before d, and to $\bar{n}_2 = \bar{n} - \bar{n}_1$ positions after d. Suppose $n_1 < \bar{n}_1$. We then transfer the job occupying the n_2th tardy position in σ (the first tardy job) to the $(n_1 + 1)$th early position. The latter position is in the optimal set; the former is not. Hence, this transfer reduces the objective value, thereby contradicting the optimality of σ. If $n_1 > \bar{n}_1$, then a similar argument applies. □

The common due date lower bound can only be used in conjunction with Theorem 6.2 if the lower bound is independent from the partial sequence $j\pi$. It is effective if the due dates are close to each other.

6.4.3.2. The common slack time problem

Consider the special case of the $1||\alpha\Sigma C_j + \beta\Sigma E_j$ problem where all jobs have equal slack time s; i.e., $d_j - p_j = s$ for each J_j $(j = 1, \ldots, n)$. This problem has the same features as the common due date problem. It is $\mathcal{NP}$-hard, unless the machine is continuously available from time $s + p_{\max} - \Sigma_{j=1}^{n} p_j$ onwards, where $p_{\max} = \max_{1 \leq j \leq n} p_j$. However, applying Lagrangian relaxation as described in the previous subsection, we derive a strong lower bound. Furthermore, the best lower bound is also computed in $O(n \min\{\alpha, n\})$ time; there are the same options to implement the lower bound. The common slack time lower bound is effective if all slack times are close to each other.

6.4.4. Fourth method: relax the processing times

Again, we consider a special case of the $1||\alpha\Sigma C_j + \beta\Sigma E_j$ problem. Assume that all processing times are equal. Theorem 6.3 indicates that the *earliest-due-date* sequence (i.e., the sequence with the jobs in order of non-decreasing due dates) is optimal. This special case is solved in $O(n^2)$ time, which is needed to compute the optimal schedule for a given sequence.

Let us return to our original problem. Define $p_{\min} = \min_{1 \leqslant j \leqslant n} p_j$. The optimal solution value of the relaxed problem $1|p_j = p_{\min}|\alpha\Sigma C_j + \beta\Sigma E_j$ provides a lower bound for the original problem: each set of job completion times that is feasible for the original problem is also feasible for the relaxed problem and has equal cost.

Given a partial schedule π, let σ be the earliest-due-date sequence for the jobs in $\mathcal{J} - \pi$, and let $g(\sigma)$ be the optimal solution value for the relaxed problem. Disregarding π, we get the lower bound $f(\pi) + g(\sigma)$. We can marginally improve on this lower bound. Suppose we have reindexed in order of non-decreasing due dates. Corollary 6.4 indicates that J_n is also scheduled last if we put its processing time equal to $\min\{p_n, p_{\min} + d_n - d_{n-1}\}$. An improved lower bound is therefore given by $f(\pi) + g(\sigma) + \alpha[\min\{p_n, p_{\min} + d_n - d_{n-1}\} - p_{\min}]$.

If the execution of jobs in σ overlap with the execution of jobs in π, then it pays to take π into regard. The lower bound is then equal to the cost for the sequence $\sigma\pi$ with the jobs in π still having their original processing times.

Both bounds are computed in $O(n^2)$ time and dominate the lower bound $\alpha\Sigma_{j=1}^{n} d_j$. Only the first version can be used in conjunction with Theorem 6.2. The common processing time lower bounds are only effective if the processing times are close to each other.

In our 3-job example, we have $p_{\min} = 3$, $d_1 = 15$, and $d_2 = d_3 = 10$. An optimal schedule for the common processing time problem is depicted in Figure 6.4. Its objective value is 39α; this is equal to the optimal solution value for the original problem.

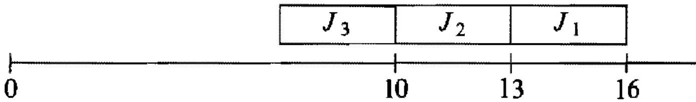


FIGURE 6.4. Optimal schedule for the common processing time problem.

6.4.5. Fifth method: Lagrangian relaxation

The problem of minimizing total inventory cost, referred to as problem (P), can be formulated as follows. Determine values C_j and E_j $(j = 1, \ldots, n)$ that minimize

$$\alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} E_j \tag{P}$$

subject to

$$E_j \geqslant 0, \qquad \text{for } j = 1, \ldots, n, \tag{6.5}$$

$$E_j \geqslant d_j - C_j, \qquad \text{for } j = 1, \ldots, n, \tag{6.6}$$

$$C_j \geqslant C_k + p_j \text{ or } C_k \geqslant C_j + p_k, \qquad \text{for } j, k = 1, \ldots, n, j \neq k, \tag{6.7}$$

$$C_j - p_j \geqslant 0, \qquad \text{for } j = 1, \ldots, n. \tag{6.8}$$

The conditions (6.5) and (6.6) reflect the definition of job earliness, while the

conditions (6.7) ensure that the machine executes at most one job at a time. The conditions (6.8) express that the machine is available only from time 0 onwards. We note that the above formulation matches the generic formulation presented in Chapter 2.

We introduce a non-negative vector $\lambda = (\lambda_1, \ldots, \lambda_n)$ of Lagrangian multipliers in order to dualize the conditions (6.5). For a given vector $\lambda \geqslant 0$, the Lagrangian problem is to determine the value $L(\lambda)$, which is the minimum of

$$\alpha \sum_{j=1}^{n} C_j + \sum_{j=1}^{n} (\beta - \lambda_j) E_j$$

subject to the conditions (6.6), (6.7), and (6.8). We know that for any given $\lambda \geqslant 0$ the value $L(\lambda)$ provides a lower bound to problem (P). If $\beta - \lambda_j < 0$ for some J_j, we get $E_j = \infty$, which disqualifies the lower bound. We therefore assume that

$$\lambda_j \leqslant \beta, \qquad \text{for } j = 1, \ldots, n. \tag{6.9}$$

This, in turn, implies that, for any solution to the Lagrangian problem, conditions (6.6) hold with equality: $E_j = d_j - C_j$ for each j ($j = 1, \ldots, n$). Hence, the Lagrangian problem, referred to as problem (L_λ), transforms into the problem of minimizing

$$\sum_{j=1}^{n} (\alpha - \beta + \lambda_j) C_j + \sum_{j=1}^{n} (\beta - \lambda_j) d_j \tag{L_λ}$$

subject to

$$C_j \geqslant C_k + p_j \text{ or } C_k \geqslant C_j + p_k, \qquad \text{for } j, k = 1, \ldots, n, j \neq k, \tag{6.7}$$

$$C_j - p_j \geqslant 0, \qquad \text{for } j = 1, \ldots, n. \tag{6.8}$$

If $\alpha - \beta + \lambda_j < 0$ for some J_j, we get $C_j = \infty$, which makes the lower bound rather weak. However, as demonstrated at the beginning of Section 6.4, we can determine an upper bound T on the job completion times, which implies that

$$C_j \leqslant T, \qquad \text{for } j = 1, \ldots, n. \tag{6.10}$$

Although the conditions (6.10) are redundant for the primal problem (P), they are essential to admit values $\lambda_j < \beta - \alpha$. For solving problem ($L_\lambda$) under these additional conditions, we first determine the sets of jobs $\mathcal{J}^+ = \{J_j \mid \lambda_j > \beta - \alpha\}$, $\mathcal{J}^- = \{J_j \mid \lambda_j < \beta - \alpha\}$, and $\mathcal{J}^0 = \{J_j \mid \lambda_j = \beta - \alpha\}$. The following theorem stipulates that problem (L_λ) is solved by a simple extension of Smith's rule [Smith, 1956] for solving the $1 \mid \mid \Sigma w_j C_j$ problem; the proof proceeds by an elementary interchange argument (see Theorem 1.1).

THEOREM 6.9. *Problem* (L_λ) *with the additional conditions* (6.10) *is solved by scheduling the jobs in* $\mathcal{J}^+$ *in non-increasing order of ratios* $(\alpha - \beta + \lambda_j)/p_j$ *in the interval* $[0, p(\mathcal{J}^+)]$, *and scheduling the jobs in* $\mathcal{J}^-$ *in non-increasing order of ratios* $(\alpha - \beta + \lambda_j)/p_j$ *in the interval* $[T - p(\mathcal{J}^-), T]$. *The remaining jobs can be scheduled in any order in the interval* $[p(\mathcal{J}^+), T - p(\mathcal{J}^-)]$. □

We are interested in determining the vector $\lambda^* = (\lambda_1^*, \ldots, \lambda_n^*)$ of Lagrangian multipliers that induces the best Lagrangian lower bound. The vector λ^* stems from solving the *Lagrangian dual problem*, referred to as problem (D): maximize

$$L(\lambda) \tag{D}$$

subject to

$$0 \leqslant \lambda_j \leqslant \beta, \quad \text{for } j = 1, \ldots, n.$$

Problem (D) is solvable to optimality in polynomial time by use of the ellipsoid method (see Theorems 1.6 and 2.1). Since the ellipsoid method is very slow in practice, we take our resort to an approximation algorithm for problem (D). We develop an ascent direction algorithm that is similar to the one developed in Section 2.1 for the $1|prec|\Sigma w_j C_j$ problem.

First, we identify the primitive directional derivatives. In the solution to the Lagrangian problem (L_λ), the position of J_j depends on the ratio $(\alpha - \beta + \lambda_j)/p_j$; we call this ratio the *relative weight* of J_j. The larger this relative weight, the smaller the completion time of J_j. If other jobs have precisely the same relative weight as J_j, then the exact position of J_j is determined by settling ties. Let now $C_j^+(\lambda)$ denote the earliest possible completion time of J_j in an optimal schedule for problem (L_λ); let $C_j^-(\lambda)$ denote the latest possible completion time of J_j in an optimal schedule for problem (L_λ). If we increase λ_j by $\epsilon > 0$, then we can choose ϵ small enough to make sure that at least one optimal schedule for problem (L_λ) remains optimal (see Theorem 1.7). In fact, all such optimal schedules must have J_j completed on time $C_j^+(\lambda)$. If we increase λ_j by such a sufficiently small $\epsilon > 0$, then the Lagrangian objective value is affected by $\epsilon(C_j^+(\lambda) - d_j)$. The primitive directional derivative for increasing λ_j, denoted by $l_j^+(\lambda)$, is therefore simply

$$l_j^+(\lambda) = C_j^+(\lambda) - d_j, \quad \text{for } j = 1, \ldots, n.$$

Hence, if $l_j^+(\lambda) > 0$, then increasing λ_j is an ascent direction. In a similar fashion, we derive that the primitive directional derivative for decreasing λ_j, denoted by $l_j^-(\lambda)$, is

$$l_j^-(\lambda) = d_j - C_j^-(\lambda), \quad \text{for } j = 1, \ldots, n.$$

If $l_j^-(\lambda) > 0$, then decreasing λ_j is an ascent direction.

Second, we determine an appropriate step size $\Delta > 0$ to move by along a chosen ascent direction. We compute the step size that takes us to the first point where the corresponding primitive directional derivative is no longer positive. If no such point exists, then we choose the step size as large as possible.

Suppose $l_j^+(\lambda) > 0$: J_j is tardy in any optimal schedule for problem (L_λ). Increasing λ_j, thereby putting J_j earlier in the schedule, is an ascent direction. We distinguish the cases $p_j - d_j > 0$, $p_j - d_j = 0$, and $p_j - d_j < 0$. Consider the case $p_j - d_j > 0$. Hence, J_j is unavoidably tardy, and $l_j^+(\lambda) > 0$ for all $\lambda \geqslant 0$ with $\lambda_j < \beta$; recall that some primitive directional derivatives do not exist at the boundaries. Therefore, we take the step size $\Delta = \beta - \lambda_j$. Accordingly, we must also have that $\lambda_j^* = \beta$; otherwise, increasing λ_j^* would be an ascent direction. If $p_j = d_j$, then there exists an optimal solution to problem (D) with $\lambda_j^* = \beta$. Find

$\mathcal{T} = \{J_j \mid p_j \geqslant d_j\}$. We have proved the following result.

THEOREM 6.10. *There exists an optimal solution for the Lagrangian dual problem* (D) *with* $\lambda_j^* = \beta$ *for each* $J_j \in \mathcal{T}$. □

Suppose now $p_j < d_j$. The step size Δ must satisfy $\lambda_j + \Delta \leqslant \beta$. We identify the first job in the schedule, say, J_k, for which $C_k - p_k + p_j \leqslant d_j$. Since $p_j < d_j$, such a J_k always exists. If J_j is scheduled in J_k's position, then J_j is not tardy. Hence, if there were no upper bound on λ, then increasing λ_j would be an ascent direction up to the point where the relative weight of J_j becomes equal to the relative weight of J_k. Hence, the maximum step size along this ascent direction is the largest value Δ such that

$$(\alpha-\beta+\lambda_j + \Delta)/p_j \leqslant (\alpha-\beta+\lambda_k)/p_k, \text{ and}$$

$$\lambda_j + \Delta \leqslant \beta.$$

Let now $\bar{\lambda} = (\lambda_1, \ldots, \lambda_j + \Delta, \ldots, \lambda_n)$. Suppose $\bar{\lambda}_j + \Delta < \beta_j$. Since the relative weights for all jobs but J_j have remained the same, optimal solutions for the problems ($L_{\bar{\lambda}}$) and (L_λ) exist with the same jobs scheduled before J_k. J_j and J_k have now equal relative weights: in any optimal solution to problem ($L_{\bar{\lambda}}$), J_j can be scheduled before J_k or after J_k. If J_j is scheduled before J_k, then J_j is not tardy; if J_j is scheduled after J_k, then J_j is not early. Hence, we have that $C_j^+(\lambda) \leqslant d_j \leqslant C_j^-(\bar{\lambda})$; the step size Δ has taken us to the first point where the primitive directional derivative for increasing λ_j is no longer positive. If $\bar{\lambda}_j = \beta$, then the step size has been chosen as large as possible.

Suppose now $l_j^-(\lambda) < 0$: J_j is early in any optimal schedule for problem (L_λ). Decreasing λ_j, thereby delaying J_j, is an ascent direction. We distinguish the cases $d_j > T$, $d_j = T$, and $d_j < T$. Consider the case $d_j > T$; hence, J_j is unavoidably early, and $l_j^-(\lambda) > 0$ for all λ with $\lambda_j > 0$. Therefore, we choose the step size as large as possible: $\Delta = \lambda_j$. Accordingly, we also must have that $\lambda_j^* = 0$; otherwise, decreasing λ_j^* would be an ascent direction. If $d_j = T$, then there exists an optimal schedule to problem (D) with $\lambda_j^* = 0$. Identify $\mathcal{E} = \{J_j \mid d_j \geqslant T\}$. We have proved the following result.

THEOREM 6.11. *There exists an optimal solution for the Lagrangian dual problem* (D) *with* $\lambda_j^* = 0$ *for each* $J_j \in \mathcal{E}$. □

Consider now the case $d_j < T$. The procedure to compute the appropriate step size Δ proceeds in a similar fashion as above. We identify some J_k as the first job in the schedule with $C_k \geqslant d_j$. If J_j is scheduled in J_k's position, then J_j is not early. Hence, if there were no lower bound on λ, then decreasing λ_j would be an ascent direction up to the point where the relative weight of J_j becomes equal to the relative weight of J_k. Hence, the maximum step size along this ascent direction is the largest value Δ for which

$$(\alpha-\beta+\lambda_j - \Delta)/p_j \geqslant (\alpha-\beta+\lambda_k)/p_k, \text{ and}$$

$$\lambda_j - \Delta \geqslant 0.$$

Let $\bar{\lambda} = (\lambda_1, \ldots, \lambda_j - \Delta, \ldots, \lambda_n)$. Suppose $\bar{\lambda}_j > 0$. Since the relative weights for all jobs but J_j have remained the same, optimal solutions for the problems $(L_{\bar{\lambda}})$ and (L_λ) exist with the same jobs scheduled after J_k. Since J_j and J_k have now equal weights, J_j can be scheduled after J_k or before J_k in any optimal schedule for problem $(L_{\bar{\lambda}})$. If J_j is scheduled after J_k, then J_j is not early; if J_j is scheduled before J_k, then J_j is not tardy. Hence, we find that $C_j^+(\bar{\lambda}) \leqslant d_j \leqslant C_j^-(\bar{\lambda})$. If $\bar{\lambda}_j = 0$, then the step was taken as large as possible.

Termination of the ascent direction procedure occurs at some $\bar{\lambda}$ where all existing primitive directional derivatives are non-positive. If all primitive directional derivatives exist at such a $\bar{\lambda}$, we have

$$C_j^+(\bar{\lambda}) \leqslant d_j \leqslant C_j^-(\bar{\lambda}), \quad \text{for } j = 1, \ldots, n.$$

These termination conditions also apply to λ^*, since they are necessary for optimality. Before implementing the ascent direction algorithm, we make use of this fact to decompose the Lagrangian dual problem (D) into two subproblems. This decomposition is achieved by partitioning $\mathcal{J}$ into four subsets, including the sets $\mathcal{T}$ and $\mathcal{E}$ we already identified.

Consider some job $J_j \in \mathcal{J} - \mathcal{E}$ with $d_j > p(\mathcal{J} - \mathcal{E})$. If $\lambda_j > \beta - \alpha$, then J_j will be early in any optimal solution to problem (L_λ). This means that $l_j^-(\lambda) > 0$, and hence we must have that $0 \leqslant \lambda_j^* \leqslant \beta - \alpha$. The set of jobs $\mathcal{F}$ that share this property is determined by the following procedure.

PARTITIONING ALGORITHM 1

Step 0. $\mathcal{F} \leftarrow \varnothing$, and reindex the jobs in $\mathcal{J} - \mathcal{E}$ according to non-increasing due dates. Let $k \leftarrow 1$.

Step 1. If $k > n - |\mathcal{E}|$ or if $d_k < p(\mathcal{J} - \mathcal{E} - \mathcal{F})$, then stop. Else $\mathcal{F} \leftarrow \mathcal{F} \cup \{J_k\}$.

Step 2. Set $k \leftarrow k + 1$; go to Step 1.

Suppose some job $J_j \in \mathcal{F}$ exists with $d_j > T - p(\mathcal{E})$. If we let $\lambda_j = \beta - \alpha$, then $C_j^-(\lambda) < d_j$; hence, decreasing λ_j is an ascent direction. Decreasing λ_j gives $(\alpha - \beta + \lambda_j)/p_j < 0$, as a result of which the execution of J_j interferes with the execution of the jobs in $\mathcal{E}$. We now partition the set $\mathcal{F}$ into subsets $\mathcal{F}_1$ and $\mathcal{F}_2$ $(\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2)$ such that $d_j \leqslant T - p(\mathcal{E} \cup \mathcal{F}_2)$ for each $J_j \in \mathcal{F}_1$, and such that $d_j > T - p(\mathcal{E} \cup \mathcal{F}_2)$ for each $J_j \in \mathcal{F}_2$. To achieve this, we use the following partitioning procedure; it is similar to the first.

PARTITIONING ALGORITHM 2

Step 0. Put $\mathcal{F}_2 \leftarrow \varnothing$, let $P \leftarrow T - p(\mathcal{E})$, and reindex the jobs in $\mathcal{F}$ according to non-increasing due dates. Let $k \leftarrow 1$.

Step 1. If $k > |\mathcal{F}|$, then stop. If $d_k \leqslant P$, then let $\mathcal{F}_1 \leftarrow \{J_k, \ldots, J_{|\mathcal{F}|}\}$, and stop. Otherwise, $\mathcal{F}_2 \leftarrow \mathcal{F}_2 \cup \{J_k\}$, and set $P \leftarrow P - p_k$.

Step 2. Set $k \leftarrow k + 1$; go to Step 1.

Let $\mathfrak{R} = \mathfrak{J} - \mathfrak{T} - \mathfrak{S} - \mathfrak{F}$.

THEOREM 6.12. *For each $J_j \in \mathfrak{F}_1$, we have that $\lambda_j^* = \beta - \alpha$.*

PROOF. Since we have $p(\mathfrak{T} \cup \mathfrak{R}) \leqslant d_j \leqslant T - p(\mathfrak{S} \cup \mathfrak{F}_2)$, the result follows. □

At this stage, we can decompose the Lagrangian dual problem (D) into two subproblems. Since $(\alpha - \beta + \lambda_j^*)/p_j = 0$ for each $J_j \in \mathfrak{F}_1$, the jobs in $\mathfrak{F}_1$ do not interfere with the execution of the other jobs. However, $\mathfrak{T}$ and $\mathfrak{R}$ interfere with each other, and $\mathfrak{S}$ and $\mathfrak{F}_2$ interfere with each other. On the one hand, we have the dual problem restricted to the sets $\mathfrak{T}$ and $\mathfrak{R}$; on the other hand, we have the dual problem restricted to the sets $\mathfrak{F}_2$ and $\mathfrak{S}$. In each optimal schedule for problem (D), the jobs in $\mathfrak{T}$ and $\mathfrak{R}$ are scheduled in the interval $[0, p(\mathfrak{T} \cup \mathfrak{R})]$, and the jobs in $\mathfrak{F}$ and $\mathfrak{S}$ are scheduled in the interval $[T - p(\mathfrak{S} \cup \mathfrak{F}_2), T]$. We give step-wise descriptions of the ascent direction algorithms for these two subproblems. Both are based upon the primitive directional derivatives and the step sizes we discussed earlier. The jobs in $\mathfrak{F}_1$ are scheduled somewhere in the interval $[p(\mathfrak{T} \cup \mathfrak{R}), T - p(\mathfrak{S} \cup \mathfrak{F}_2)]$; they are left out of consideration. We introduce some new notation. Let $(L_\lambda^{\mathfrak{R} \cup \mathfrak{T}})$ and $(L_\lambda^{\mathfrak{S} \cup \mathfrak{F}_2})$ denote the Lagrangian problem restricted to the set $\mathfrak{R} \cup \mathfrak{T}$ and to the set $\mathfrak{S} \cup \mathfrak{F}_2$; let $L^{\mathfrak{R} \cup \mathfrak{T}}(\lambda)$ and $L^{\mathfrak{S} \cup \mathfrak{F}_2}(\lambda)$ denote their optimal solution values.

ASCENT DIRECTION ALGORITHM FOR THE SET $\mathfrak{T} \cup \mathfrak{R}$

Step 0. For each $J_j \in \mathfrak{T}$, set $\lambda_j \leftarrow \lambda_j^* = \beta$; for each $J_j \in \mathfrak{R}$, set $\lambda_j \leftarrow \beta$. Solve $(L_\lambda^{\mathfrak{R} \cup \mathfrak{T}})$, settling ties arbitrarily; compute the job completion times.

Step 1. For each $J_j \in \mathfrak{R}$, do the following:

(a) If $C_j^-(\lambda) < d_j$, identify J_k as the first job in the schedule with $C_k \geqslant d_j$. Compute the largest value Δ such that

$$(\alpha - \beta + \lambda_j - \Delta)/p_j \geqslant (\alpha - \beta + \lambda_k)/p_k, \text{ and} \tag{6.11}$$

$$\lambda_j - \Delta \geqslant \beta - \alpha. \tag{6.12}$$

Decrease λ_j by Δ, reposition J_j according to its new relative weight, and update the job completion times.

(b) If $C_j^+(\lambda) > d_j$, identify J_k that is the first job in the schedule with $C_k - p_k + p_j \leqslant d_j$. Compute the largest value for Δ such that

$$(\alpha - \beta + \lambda_j + \Delta)/p_j = (\alpha - \beta + \lambda_k)/p_k, \text{ and}$$

$$\lambda_j + \Delta \leqslant \beta.$$

Increase λ_j by Δ, reposition J_j according to its new relative weight, and update the job completion times.

Step 2. If no multiplier adjustment has taken place, then compute $L^{\mathfrak{R} \cup \mathfrak{T}}(\lambda)$ and stop. Otherwise, go to Step 1.

THEOREM 6.13. *The procedure described above generates a series of monotonically increasing values $L^{\mathfrak{R} \cup \mathfrak{T}}(\lambda)$.*

PROOF. The proof proceeds in the same spirit as the proof of Theorem 2.2. First, consider some $J_j \in \mathfrak{R}$ with $C_j^-(\lambda) < d_j$: decreasing λ_j is an ascent direction. For brevity, we let $\mu_j = \alpha - \beta + \lambda_j$ for each j ($j = 1, \ldots, |\mathfrak{R}\cup\mathfrak{T}|$). We reindex the jobs in order of non-increasing values μ_j / p_j, settling all ties arbitrarily except for J_j: we give J_j the largest index possible. Accordingly, we obtain the sequence $(J_1, \ldots, J_{|\mathfrak{R}\cup\mathfrak{T}|})$, which is optimal for problem $(L_\lambda^{\mathfrak{R}\cup\mathfrak{T}})$, with job completion times $C_1, \ldots, C_{|\mathfrak{R}\cup\mathfrak{T}|}$. We note that $C_j = C_j^-(\lambda)$. Let Δ be the step size computed as prescribed in the ascent direction algorithm, and let $\bar{\lambda} = (\lambda_1, \ldots, \lambda_j - \Delta_1, \ldots, \lambda_{|\mathfrak{R}\cup\mathfrak{T}|})$.

We distinguish the case that condition (6.11) holds with equality from the case that condition (6.12) holds with equality. Consider the first case; accordingly let J_k be the job specified in the ascent direction procedure. In more detail, the sequence under consideration is $(J_1, \ldots, J_{j-1}, J_j, J_{j+1}, \ldots, J_{k-1}, J_k, J_{k+1}, \ldots, J_{|\mathfrak{R}\cup\mathfrak{T}|})$; an optimal sequence for problem $(L_\lambda^{\mathfrak{R}\cup\mathfrak{T}})$ is then $(J_1, \ldots, J_{j-1}, J_{j+1}, \ldots, J_k, J_j, J_{k+1}, \ldots, J_{|\mathfrak{R}\cup\mathfrak{T}|})$. The job completion times for the latter sequence can conveniently be expressed in terms of $C_1, \ldots, C_{|\mathfrak{R}\cup\mathfrak{T}|}$. We now prove that $L^{\mathfrak{R}\cup\mathfrak{T}}(\bar{\lambda}) > L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda)$. We have

$$\begin{aligned} L^{\mathfrak{R}\cup\mathfrak{T}}(\bar{\lambda}) &= \sum_{i=1}^{j-1} \mu_i C_i + (\mu_j - \Delta)(C_j^-(\lambda) + \sum_{i=j+1}^{k} p_i) + \\ &\quad \sum_{i=j+1}^{k} \mu_i (C_i - p_j) + \sum_{i=k+1}^{|\mathfrak{R}\cup\mathfrak{T}|} \mu_i C_i + \sum_{i=1}^{|\mathfrak{R}\cup\mathfrak{T}|} (\beta - \lambda_i) d_i + \Delta d_j \\ &= L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda) - p_j \sum_{i=j+1}^{k} \mu_i + \mu_j \sum_{i=j+1}^{k} p_i - \Delta(C_j^-(\lambda) + \sum_{i=j+1}^{k} p_i - d_j) \\ &= L(\lambda) - p_j \sum_{i=j+1}^{k-1} \mu_i + \mu_j \sum_{i=j+1}^{k-1} p_i - \Delta(C_j^-(\lambda) + \sum_{i=j+1}^{k-1} p_i - d_j) + \\ &\quad (\mu_j - \Delta) p_k - p_j \mu_k. \end{aligned}$$

Note that $(\mu_j - \Delta)/p_j = \mu_k / p_k$; hence, we have $(\mu_j - \Delta)p_k - p_j \mu_k = 0$. This implies that

$$L(\bar{\lambda}) \geqslant L(\lambda) + p_j \sum_{i=j+1}^{k-1} \left[p_i(\mu_j/p_j - \mu_i/p_i) \right] - \Delta(C_j^-(\lambda) + \sum_{i=j+1}^{k-1} p_i - d_j).$$

Since $d_j > C_j^-(\lambda) + \sum_{i=j+1}^{k-1} p_i$, $\mu_j / p_j > \mu_i / p_i$ for each i ($i = j+1, \ldots, k-1$), and $\Delta > 0$, we have that $L^{\mathfrak{R}\cup\mathfrak{T}}(\bar{\lambda}) > L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda)$.

Now assume that the condition (6.12) holds with equality and the condition (6.11) does not: $\Delta = \alpha - \beta + \lambda_j$. This implies that J_j will now be placed after some job J_h, with $j \leqslant h < k$. For this case, the second sequence is $(J_1, \ldots, J_{j-1}, J_{j+1}, \ldots, J_h, J_j, J_{h+1}, \ldots, J_k, \ldots, J_{|\mathfrak{R}\cup\mathfrak{T}|})$. We perform a similar analysis as above to obtain

$$L^{\mathfrak{R}\cup\mathfrak{T}}(\bar{\lambda}) = L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda) - p_j \sum_{i=j+1}^{h} \mu_i + \mu_j \sum_{i=j+1}^{h} p_i - \Delta(C_j^-(\lambda) + \sum_{i=j+1}^{h} p_i - d_j) =$$

$$= L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda) + p_j \sum_{i=j+1}^{h} \left[p_i(\mu_j/p_j - \mu_i/p_i)\right] - \Delta(C_j^-(\lambda) + \sum_{i=j+1}^{h} p_i - d_j).$$

At this point, similar arguments as before apply to show that $L^{\mathfrak{R}\cup\mathfrak{T}}(\bar{\lambda}) > L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda)$.

Second, consider the case that $C_j^+(\lambda) > d_j$ for some $J_j \in \mathfrak{R}$: increasing λ_j is an ascent direction. Let Δ be the desired step size, computed as described in the ascent direction algorithm. The proof to show that $L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda_1, \ldots, \lambda_j + \Delta, \ldots, \lambda_{|\mathfrak{R}\cup\mathfrak{T}|}) > L^{\mathfrak{R}\cup\mathfrak{T}}(\lambda_1, \ldots, \lambda_j, \ldots, \lambda_{|\mathfrak{R}\cup\mathfrak{T}|})$ follows the same lines as above. □

Ascent Direction Algorithm for the Set $\mathfrak{F}_2 \cup \mathfrak{E}$

Step 0. Set $\lambda_j \leftarrow \beta - \alpha$ for each $J_j \in \mathfrak{F}_2$, and $\lambda_j \leftarrow \lambda_j^* = 0$ for each $J_j \in \mathfrak{E}$. Solve $(L_\lambda^{\mathfrak{E}\cup\mathfrak{F}_2})$, settling ties arbitrarily; compute the job completion times.

Step 1. For each $J_j \in \mathfrak{F}_2$, do the following:

(a) If $C_j^-(\lambda) < d_j$, identify J_k as the first job in the schedule with $C_k \geq d_j$. Compute the largest value Δ such that

$$(\alpha - \beta + \lambda_j - \Delta)/p_j \geq (\alpha - \beta + \lambda_k)/p_k, \text{ and}$$
$$\Delta \leq \lambda_j.$$

Decrease λ_j by Δ, reposition J_j according to its new relative weight, and update the job completion times.

(b) If $C_j^+(\lambda) > d_j$, identify J_k that is the first job in the schedule with $C_k \leq d_j + p_k - p_j$. Compute the largest value for Δ such that

$$(\alpha - \beta + \lambda_j + \Delta)/p_j = (\alpha - \beta + \lambda_k)/p_k, \text{ and}$$
$$\lambda_j + \Delta \leq \beta - \alpha.$$

Increase λ_j by Δ, reposition J_j according to its new relative weight, and update the job completion times.

Step 2. If no multiplier adjustment has taken place, then compute $L^{\mathfrak{R}\cup\mathfrak{T}}(\bar{\lambda})$ and stop. Otherwise, go to Step 1.

Theorem 6.14. *The procedure described above generates a series of monotonically increasing values* $L^{\mathfrak{R}\cup\mathfrak{T}}(\bar{\lambda})$.

Proof. The proof proceeds along the same lines as the proof of Theorem 6.13. □

For each $J_j \in \mathfrak{J} - \mathfrak{F}_1$, let C_j and $\bar{\lambda}_j$ denote the completion time and the Lagrangian multiplier upon termination of the appropriate ascent direction algorithm. We note that $\bar{\lambda}_j = \beta_j$ for each $J_j \in \mathfrak{T}$, $\bar{\lambda}_j = \beta - \alpha$ for each $J_j \in \mathfrak{F}_1$, and $\bar{\lambda}_j = 0$ for each $J_j \in \mathfrak{E}$. Hence, the overall Lagrangian lower bound is given by

$$L(\bar{\lambda}) = \sum_{J_j \in \mathfrak{T}} \alpha C_j + \sum_{J_j \in \mathfrak{F}_1} \alpha d_j + \sum_{J_j \in \mathfrak{E}} \left[(\alpha - \beta)C_j + \beta d_j\right] +$$

$$+ \sum_{J_j \in \mathfrak{R} \cup \mathfrak{T}_2} \left[(\alpha - \beta + \bar{\lambda}) C_j - (\beta - \bar{\lambda}_j) d_j \right]$$

6.5. Computational results

The algorithm was coded in the computer language C; the experiments were conducted on a Compaq-386/20 Personal Computer. The algorithm was tested on instances with 8, 10, 12, 15, and 25 jobs. The processing times were generated from the uniform distribution [10,100]. The due dates were generated from the uniform distribution $[P(1-T-R/2), P(1-T+R/2)]$, where $P = \Sigma_{j=1}^{n} p_j$ and where R and T are parameters. For both parameters, we considered the values 0.2, 0.4, 0.6, 0.8, and 1.0. This procedure to generate due dates parallels the procedure described by Potts and Van Wassenhove [1985] for the weighted tardiness problem. For each combination of T, P, and n, we generated 5 instances. Each instance was considered with $\alpha = 1$ and with β running from 2 to 5.

The general impression was that instances become difficult with smaller values of T, with smaller values of R, and with smaller values of β. A small value of T induces relative large due dates, implying that the machine will be idle for some time before processing the first job. A small value of R induces due dates that are close to each other; it is then harder to partition the jobs. A large value of β implies that earliness is severely penalized; most jobs will therefore be tardy. Accordingly, the instances with $T = 0.2$, $R = 0.2$, and $\beta = 5$ are the hardest; the instances with $T = 1.0$, $R = 1.0$, and $\beta = 2$ are the easiest.

Table 6.2 exhibits a summary of our computational results; we only report the results for the instances with T and R equal. It shows that instances with up to 10 jobs are easy. For $n = 12$, the instances with $T = R = 0.2$ require already considerable effort. For $n = 20$, only the choice $T = R = 1.0$ induces instances that are solvable within reasonable time limits. It is likely, however, that the performance of the algorithm is considerably enhanced by fine-tuning the algorithm to specific instances. Currently, all lower bounds are computed in each node of the tree; Lagrangian relaxation, for instance, is useless for instances with $T = R = 0.2$.

6.6. Conclusions

Although machine idle time is a practical instrument to reduce inventory cost, a considerable lack of theoretical analysis of related machine scheduling problems exists. Within this context, we have addressed the $1 \| \alpha \Sigma C_j + \beta \Sigma E_j$ problem for the case that $\alpha < \beta$. It is a very difficult problem from a practical point of view.

n	T,R	$\beta=2$ nodes	sec	$\beta=3$ nodes	sec	$\beta=4$ nodes	sec	$\beta=5$ nodes	sec
8	0.2	417	2	406	2	301	2	58	1
8	0.4	131	1	198	1	185	1	31	1
8	0.6	34	1	48	1	29	1	5	1
8	0.8	23	1	37	1	14	1	8	1
8	1.0	20	1	36	1	33	1	15	1
10	0.2	2438	8	2525	9	2088	7	484	2
10	0.4	266	2	689	3	570	3	202	2
10	0.6	123	1	110	1	88	1	52	1
10	0.8	126	1	122	1	107	1	64	1
10	1.0	109	1	140	1	78	1	40	1
12	0.2	30182	103	26676	106	18358	78	10487	48
12	0.4	15176	66	20756	100	15613	75	10391	50
12	0.6	212	2	262	2	53	1	10	1
12	0.8	380	2	576	4	300	2	170	1
12	1.0	432	2	527	3	226	2	96	1
15	0.2	-	-	-	-	-	-	(2)	-
15	0.4	(3)	-	(2)	-	(2)	-	(30	-
15	0.6	1414	10	2407	17	927	7	339	2
15	0.8	1665	13	1865	15	1647	14	540	5
15	1.0	493	6	402	17	2063	17	1082	9
20	0.2	-	-	-	-	-	-	-	-
20	0.4	-	-	-	-	-	-	-	-
20	0.6	7991	80	13169	136	5529	62	2048	24
20	0.8	8183	85	7244	84	4016	55	1318	21
20	1.0	5127	49	5243	41	2191	32	651	12

TABLE 6.2. Computational results. For each combination of n ($n=8,10,12,15,20$), of T and R ($T=R=0.2,0.4,0.6,0.8,1.0$), and of β ($\beta=2,3,4,5$), we present the average number of nodes and the average number of seconds; the average was computed over 5 instances. All averages were rounded up to the nearest integer. The sign '-' indicates that not all instances of this particular combination could be solved without examining more than 100,000 nodes.

References

I. ADIRI AND N. AMIT (1984). Openshop and flowshop scheduling to minimize sum of completion times. *Computers and Operations Research 11*, 275-284.

D. ADOLPHSON AND T.C. HU (1973). Optimal linear ordering. *SIAM Journal of Applied Mathematics 25*, 403-423.

V. AGGARWAL (1985). A Lagrangian-relaxation method for the constrained assignment problem. *Computers and Operations Research 12*, 97-106.

A.V. AHO, J.E. HOPCROFT, AND J.D. ULLMAN (1982). *Data Structures and Algorithms*, Addison-Wesley, Reading, Massachusetts.

U. BAGCHI, Y.L. CHANG, AND R.S. SULLIVAN (1987). Minimizing absolute and squared deviation of completion times with different earliness and tardiness penalties and a common due date. *Naval Research Logistics 34*, 739-751.

K.R. BAKER (1974). *Introduction to Sequencing and Scheduling*, Wiley, New York.

K.R. BAKER AND G. SCUDDER (1990). Sequencing with earliness and tardiness penalties: a review. *Operations Research 38*, 22-57.

E. BALAS (1985). On the facial structure of scheduling polyhedra. *Mathematical Programming Study 24*, 179-218.

E. BALAS AND N. CHRISTOFIDES (1981). A restricted Lagrangean approach to the traveling salesman problem. *Mathematieal Programming 21*, 19-46.

E. BALAS AND P. TOTH (1985). Branch-and-bound algorithms for the traveling salesman problem. E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS (eds.). *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, Wiley, Chichester, Chapter 8, 251-306.

S.P. BANSAL (1977). Minimizing the sum of completion times of n jobs over m machines in a flowshop - a branch and bound approach. *AIIE Transactions 9*, 306-311.

P. BARCIA AND K. JÖRNSTEN (1990). Improved Lagrangian decomposition: an application to the generalized assignment problem. *European Journal of*

Operational Research 46, 84-92.

M.S. BAZARAA AND J.J. GOODE (1979). A survey of various tactics for generating Lagrangean multipliers in the context of Lagrangian duality. *European Journal of Operational Research 3*, 322-328.

R. BELLMAN, A.O. ESOGBUE, AND I. NABESHIMA (1982). *Mathematical Aspects of Scheduling & Applications*, Pergamon Press, Oxford.

P.P. BERGMANS (1972). Minimizing expected travel time on geometrical patterns by optimal probability rearrangements. *Information and Control 20*, 331-350.

M. BERRADA AND K.E. STECKE (1986). A branch-and-bound approach for machine load balancing in flexible manufacturing systems. *Management Science 32*, 1316-1335.

O. BILDE AND S. KRARUP (1977). Sharp lower bounds and efficient algorithms for the simple plant location problem. *Annals of Discrete Mathematics 1*, 79-88.

P.M. CAMERINI, L. FRATTA, AND F. MAFFIOLI (1975). On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study 3*, 26-34.

N. CHRISTOFIDES (1970). The shortest hamiltonian chain of a graph. *SIAM Journal of Applied Mathematics 19*, 689-696.

R.W. CONWAY, W.L. MAXWELL, AND L.W. MILLER (1967). *Theory of Scheduling*, Addison-Wesley, Reading, Massachusetts.

S.A. COOK (1971). The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, 151-158.

E. DAVIS AND J.M. JAFFE (1981). Algorithms for scheduling tasks on unrelated parallel processors. *Journal of the Association for Computing Machinery 28*, 721-736.

P. DE AND T.E. MORTON (1980). Scheduling to minimize makespan on unequal parallel processors. *Decision Sciences 11*, 586-603.

J. DU AND J.Y.-T. LEUNG (1990). Minimizing total tardiness on one machine is $\mathcal{NP}$-hard. *Mathematics of Operations Research 15*, 483-495.

M.E. DYER AND L.A. WOLSEY (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics 26*, 255-270.

H. EMMONS (1987). Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics 34*, 803-810.

D. ERLENKOTTER (1978). A dual-based procedure for uncapacitated facility location. *Operations Research 26*, 992-1009.

M.L. FISHER (1973). Optimal solution of scheduling problems using Lagrange multipliers: part I. *Operations Research 21*, 1114-1127.

M.L. FISHER (1976). A dual algorithm for the one-machine scheduling problem. *Mathematical Programming 11*, 229-251.

M.L. FISHER (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science 27*, 1-18.

M.L. FISHER (1985). An applications oriented guide to Lagrangian relaxation. *Interfaces 15*, 10-21.

M.L. FISHER, R. JAIKUMAR, AND L.N. VAN WASSENHOVE (1986). A multiplier

adjustment method for the generalized assignment problem. *Management Science 32*, 1098-1103

M.L. FISHER AND P. KEDIA (1990). Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science 36*, 674-688.

S. FRENCH (1982). *Sequencing and Scheduling: an Introduction to the Mathematics of the Job-Shop*, Horwood, Chichester.

T.D. FRY AND G. KEONG LEONG (1987A). Single machine scheduling: a comparison of two solution procedures. *Omega 15*, 277-282.

T.D. FRY AND G. KEONG LEONG (1987B). A bi-criterion approach to minimizing inventory costs on a single machine when early shipments are forbidden. *Computers and Operations Research 14*, 363-368.

M.R. GAREY AND D.S. JOHNSON (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

M.R. GAREY, D.S. JOHNSON, AND R. SETHI (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*, 117-129.

M.R. GAREY, R.E. TARJAN, AND G.T. WILFONG (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research 13*, 330-348.

A.M. GEOFFRION (1974A). Lagrangian relaxation and its uses in integer programming. *Mathematical Programming Study 2*, 82-114.

A.M. GEOFFRION (1974B). Duality in nonlinear programming: a simplified applications-oriented development. *SIAM Review 13*, 1-37.

F. GLOVER (1989). Tabu search - Part I. *ORSA Journal on Computing 1*, 190-206.

T. GONZALEZ, E.L. LAWLER, AND S. SAHNI (1990). Optimal preemptive scheduling of two unrelated processors. *ORSA Journal on Computing 2*, 219-224.

R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, AND A.H.G RINNOOY KAN (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, 287-326.

H.J. GREENBERG AND W.P. PIERSKALLA (1970). Surrogate mathematical programming. *Operations Research 18*, 924-939.

M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica 1*, 169-197. [Corrigendum (1984): *Combinatorica 4*, 291-295.]

M. GRÖTSCHEL AND M.W. PADBERG (1985). Polyhedral theory. E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS (eds.). *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, Wiley, Chichester, Chapter 8, 251-306.

M. GUIGNARD AND M.B. ROSENWEIN (1989). An application-oriented guide for designing Lagrangean dual ascent algorithms. *European Journal of Operational Research 43*, 197-205.

M. GUIGNARD AND M.B. ROSENWEIN (1990). An improved dual based algorithm for the generalized assignment problem. *Operations Research 37*, 658-663.

M. GUIGNARD AND K. SPIELBERG (1979). A direct dual method of the mixed plant location problem with some side constraints. *Mathematical Programming 17*, 198-228.

J.N.D. GUPTA AND R.A. DUDEK (1971). Optimality criteria for flowshop

schedules. *AIIE Transactions 3*, 199-205.

N.G. HALL, W. KUBIAK, AND S.P. SETHI (1991). Deviation of completion times about a common due date. To appear in *Operations Research*.

G. HANDLER AND I. ZANG (1980). A dual algorithm for the constrained shortest path problem. *Networks 10*, 293-310.

A.M.A. HARIRI AND C.N. POTTS (1983). An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics 5*, 99-109.

A.M.A. HARIRI AND C.N. POTTS (1984). Algorithms for two-machine flow-shop sequencing with precedence constraints. *European Journal of Operational Research 17*, 238-248.

A.M.A. HARIRI AND C.N. POTTS (1990). *Heuristics for scheduling unrelated parallel machines*, Working Paper, Faculty of Mathematics, University of Southampton.

M. HELD AND R.M. KARP (1970). The traveling salesman problem and minimum spanning trees. *Operations Research 18*, 1138-1162.

M. HELD AND R.M. KARP (1971). The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming 1*, 6-25.

M. HELD, P. WOLFE, AND H. CROWDER (1974). Validation of subgradient optimization. *Mathematical Programming 6*, 62-88.

J.A. HOOGEVEEN (1990). Minimizing maximum earliness and maximum lateness on a single machine. *Proceedings of the First Conference on Integer Programming and Combinatorial Optimization*, University of Waterloo, Waterloo, 283-295.

J.A. HOOGEVEEN, H. OOSTERHOUT, AND S.L. VAN DE VELDE (1990). *New Lower and Upper Bounds for Scheduling Around a Small Common Due Date*, Report BS-R9030, CWI, Amsterdam.

J.A. HOOGEVEEN AND S.L. VAN DE VELDE (1990). *Polynomial-Time Algorithms for Single-Machine Multicriteria Scheduling*, Report BS-R9008, CWI, Amsterdam.

J.A. HOOGEVEEN AND S.L. VAN DE VELDE (1991A). Scheduling around a small common due date. To appear in *European Journal of Operational Research*.

J.A. HOOGEVEEN AND S.L. VAN DE VELDE (1991B). *Minimizing Total Inventory Cost on a Single Machine in Just-in-Time Manufacturing*. In preparation.

J.E. HOPCROFT AND R.M. KARP (1973). An $n^{2.5}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing 2*, 225-231.

W.A. HORN (1972). Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM Journal of Applied Mathematics 23*, 189-202.

E. HOROWITZ AND S. SAHNI (1976). Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the Association for Computing Machinery 23*, 317-327.

O.H. IBARRA AND C.G. KIM (1977). On heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the Association for Computing Machinery 24*, 280-289.

E. IGNALL AND L. SCHRAGE (1965). Application of the branch and bound

technique for some flow-shop scheduling problems. *Operations Research 13*, 400-412.

S.M. JOHNSON (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly 1*, 61-68.

K. JÖRNSTEN AND M. NÄSBERG (1986). A new Lagrangian relaxation approach to the generalized assignment problems. *European Journal of Operational Research 27*, 313-323.

J.J. KANET (1981). Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly 28*, 643-651.

R.M. KARP (1972). Reducibility among combinatorial problems. R.E. MILLER AND J.W. THATCHER (eds.). *Complexity of Computer Computations*, Plenum Press, New York, 85-103.

M.H. KARWAN AND B. RAM (1987). A Lagrangean dual-based solution method for a special linear programming problem. *Computers and Operations Research 14*, 67-73.

M.H. KARWAN AND R.L. RARDIN (1979). Some relationships between Lagrangian and surrogate duality in integer programming. *Mathematical Programming 17*, 320-334.

L.G. KHACHIYAN (1979). A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR 244*, 1093-1096 (English translation: *Soviet Mathematics Doklady 20*, 191-194).

T.D. KLASTORIN (1979). An effective subgradient algorithm for the generalized assignment problem. *Computers and Operations Research 6*, 155-164.

W.H. KOHLER AND K. STEIGLITZ (1975). Exact, approximate and guaranteed accuracy algorithms for the flow-shop problem $n/2/F/\bar{F}$. *Journal of the Association for Computing Machinery 22*, 106-114.

A.W.J. KOLEN (1986). *A Polynomial Algorithm for the Linear Ordering Problem with Weights in Product Form*, Report 8622/A, Erasmus University, Rotterdam.

M.J. KRONE AND K. STEIGLITZ (1974). Heuristic programming solution of a flowshop-scheduling problem. *Operations Research 22*, 629-638.

P.J.M. VAN LAARHOVEN AND E.H.L. AARTS (1987). *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht.

E.L. LAWLER (1978). Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics 2*, 75-90.

E.L. LAWLER (1979). *Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems*, Report BW 106, CWI, Amsterdam.

E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS (eds.) (1985). *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS (1989). Sequencing and scheduling: algorithms and complexity. To appear in the *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, edited by S.C. Graves, A.H.G. Rinnooy Kan and P. Zipkin, and to be published by North-Holland.

E.L. LAWLER AND J.M. MOORE (1969). A functional equation and its application to resource allocation and sequencing problems. *Management Science 16*, 77-84.

J.K. LENSTRA AND A.H.G. RINNOOY KAN (1978). Complexity of scheduling under precedence constraints. *Operations Research 26*, 22-35.

J.K. LENSTRA, D.B. SHMOYS, AND E. TARDOS (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming 46*, 259-271.

M. MINOUX (1986). *Mathematical Programming: Theory and Algorithms*, Wiley, Chichester.

C.L. MONMA AND J.B. SIDNEY (1979). Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research 3*, 215-224.

T.E. MORTON AND B.G. DHARAN (1978). Algoristics for single-machine sequencing with precedence constraints. *Management Science 24*, 1011-1020.

G.L. NEMHAUSER AND L.A. WOLSEY (1988). *Integer and Combinatorial Optimization*, Wiley, New York.

P.S. OW AND T.E. MORTON (1989). The single machine early/tardy problem. *Management Science 35*, 177-191.

M.W. PADBERG AND M.R. RAO (1982). Odd minimum cut-sets and *b*-matchings. *Mathematics of Operations Research 7*, 67-80.

M.W. PADBERG AND G. RINALDI (1987). Optimization of a 512-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters 6*, 1-8.

C.H. PAPADIMITRIOU AND K. STEIGLITZ (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

J.-C. PICARD AND M. QUEYRANNE (1982). On the one-dimensional space allocation problem. *Operations Research 29*, 371-391.

B.T. POLYAK (1967). A general method for solving extremum problems. *Doklady Akademii Nauk SSSR 174*, 33-36 (English translation: *Soviet Mathematics Doklady 8*, 593-597).

B.T. POLYAK (1969). Minimization of unsmooth functionals. *Zurnal Vycislitelnoi Matematiki i Matematiceskoi Fiziki 9*, 509-521 (English translation: *U.S.S.R. Computational Mathematics and Mathematical Physics 9*, 14-29.

C.N. POTTS (1985A). A Lagrangean based branch-and-bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science 31*, 1300-1311.

C.N. POTTS (1985B). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics 10*, 155-164.

C.N. POTTS AND L.N. VAN WASSENHOVE (1983). An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *European Journal of Operational Research 33*, 363-377.

C.N. POTTS AND L.N. VAN WASSENHOVE (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research 33*, 363-377.

C.N. POTTS AND L.N. VAN WASSENHOVE (1988). *Single Machine Tardiness Sequencing Heuristics*, Report 8906/A, Erasmus University, Rotterdam.

V.R. PRATT (1972). An $O(n \log n)$ algorithm to distribute n records in a

sequential access file. R.E. MILLER AND J.W. THATCHER (eds.). *Complexity of Computer Computations*, Plenum Press, New York, 111-118.

M. QUEYRANNE AND Y. WANG (1988). *Single Machine Scheduling Polyhedra with Precedence Constraints*, Working Paper No. 88-MSC-017, University of British Columbia, Vancouver.

S. SARIN AND M.H. KARWAN (1987). A computational evaluation of two subgradient search methods. *Computers and Operations Research 14*, 241-247.

L. SCHRAGE AND K.R. BAKER (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research 26*, 444-449.

A. SCHRIJVER (1987). *Theory of Linear and Integer Programming*, Wiley, Chichester.

J.F. SHAPIRO (1974). A survey of Lagrangian techniques for discrete optimization. *Annals of Discrete Mathematics 5*, 113-138.

J.B. SIDNEY (1975). Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research 23*, 283-298.

W.E. SMITH (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly 3*, 59-66.

H.I. STERN (1976). *Minimizing Makespan for Independent Jobs on Nonidentical Machines - an Optimal Procedure*, Working Paper 2/75, Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva.

P.S. SUNDARARAGHAVAN AND M.U. AHMED (1984). Minimizing the sum of absolute lateness in single-machine and multimachine scheduling. *Naval Research Logistics Quarterly 31*, 325-333.

W. SZWARC (1983). The flow-shop problem with mean completion time criterion. *IIE Transactions 15*, 172-176.

W. SZWARC (1989). Single machine scheduling to minimize absolute deviation of completion times from a common due date. *Naval Research Logistics 36*, 663-673.

G.J. THOMPSON AND D.J. ZAWACK (1985/6). A problem expanding parametric method for solving the job shop scheduling problem. *Annals of Operations Research 4*, 327-342.

S.L. VAN DE VELDE (1990A). Minimizing the sum of the job completion times in the two-machine flow shop by Lagrangian relaxation. *Annals of Operations Research 26*, 257-268.

S.L. VAN DE VELDE (1990B). Dual decomposition of single machine scheduling problems. *Proceedings of the First Conference on Integer Programming and Combinatorial Optimization*, University of Waterloo, Waterloo, 495-507.

S.L. VAN DE VELDE (1990C). *Duality-Based Algorithms for Scheduling Unrelated Parallel Machines*, Report BS-R9010, CWI, Amsterdam.

D. DE WERRA AND A. HERTZ (1989). Tabu search techniques: a tutorial and an application to neural networks. *OR Spektrum 11*, 131-141.

Samenvatting

Produktieplanning en computer scheduling vormen een door de praktijk gemotiveerd onderzoeksgebied binnen de mathematische besliskunde. *Machinevolgordeproblemen* nemen daarbij een belangrijke plaats in. Dergelijke problemen betreffen het plannen van *orders* op *machines* met beperkte beschikbaarheid en capaciteit.

Een order bestaat uit een geordende lijst van operaties, die elk een van te voren vastgestelde tijd op een bepaalde machine vergen. Een machine kan niet meer dan één operatie tegelijkertijd uitvoeren en is continu beschikbaar vanaf tijdstip 0. Verder kan een order niet meer dan één operatie tegelijkertijd ondergaan. Een *plan* legt voor elke order vast wanneer en door welke machines de bijbehorende operaties uitgevoerd worden. Het streven is de produktiekosten, over het algemeen gespecificeerd als een functie van de completeringstijdstippen van de orders, te minimaliseren.

De verscheidenheid aan machineconfiguraties, eigenschappen van orders, en doelstellingsfuncties leidt tot een enorm aantal verschillende machinevolgordeproblemen. Niettemin komt ieder probleem uiteindelijk neer op het bepalen van óf een volgorde van de orders, óf een toewijzing van de orders aan machines, óf een partitie van de orders. Dit betekent dat voor ieder probleem in wezen een *eindig*, maar mogelijk enorm groot, aantal relevante oplossingen bestaat. Problemen met deze eigenschap heten *combinatorische optimaliseringsproblemen.*

Sommige problemen zijn *gemakkelijk*. Een probleem is gemakkelijk indien er een methode bestaat die een optimale oplossing vindt in een aantal basisbewerkingen (optellen, aftrekken, vermenigvuldigen, enz.) dat van boven begrensd wordt door een polynoom in de grootte van het probleem. Het probleem is dan *oplosbaar in polynomiale tijd.* De grootte van een machinevolgordeprobleem kan worden uitgedrukt in bijvoorbeeld het aantal orders en het aantal machines.

Veel machinevolgordeproblemen blijken echter *𝒩𝒫-lastig* te zijn. Als een probleem 𝒩𝒫-lastig is, dan is het zeer onwaarschijnlijk dat het probleem opgelost kan worden in polynomiale tijd. De eindigheid van de oplossingsverzameling suggereert dat *expliciete* of *volledige* aftelling van de elementen een effectieve oplossingsmethode is. Dit is bedrieglijk. Het aantal relevante oplossingen neemt over het algemeen exponentiëel toe met het aantal machines of met het aantal orders. Daarom is deze methode slechts effectief voor problemen van bescheiden omvang. Door middel van *impliciete* aftelling van de oplossingsverzameling kunnen problemen van ruimere omvang opgelost worden. *Branch-and-bound* en *dynamische programmering* zijn twee methoden die zich hierop richten. Beide methoden vergen in het slechtste geval niettemin meer dan polynomiale tijd. Branch-and-bound, bijvoorbeeld, komt in het slechtste geval neer op expliciete aftelling.

Voor 𝒩𝒫-lastige problemen staat men in wezen voor de keuze: óf men ontwikkelt een *optimaliseringsalgoritme,* die een exponentiële hoeveelheid tijd kan vergen, óf men ontwikkelt een *benaderingsalgoritme,* die minder tijd vergt, maar geen optimaliteit van de oplossing garandeert.

Lagrangiaanse relaxatie is een techniek die veel heeft bijgedragen aan de ontwikkeling van efficiëntere optimaliseringsalgoritmen voor 𝒩𝒫-lastige combinatorische problemen. Het idee achter Lagrangiaanse relaxatie is het zien van een 𝒩𝒫-lastig probleem als een gemakkelijk probleem, gecompliceerd door een aantal 'vervelende' beperkingen. Elk van deze vervelende beperkingen wordt gewogen met een niet-negatieve factor, de zogenaamde Lagrangiaanse multiplicator, en wordt vervolgens opgenomen in de doelstellingsfunctie. Voor gegeven multiplicatoren verkrijgt men aldus het *Lagrangiaanse probleem.* Dit probleem is eenvoudiger op te lossen; alle vervelende beperkingen zijn immers verwijderd. Bovendien kan men aantonen dat de optimale oplossingswaarde van het Lagrangiaanse probleem een ondergrens voor de optimale oplossingswaarde van het oorspronkelijke probleem is. Deze ondergrenzen worden gebruikt in branch-and-bound algoritmen. Het *Lagrangiaanse duale probleem* is het vinden van de Lagrangiaanse multiplicatoren die tot de beste ondergrens leiden.

Om wat voor reden dan ook is Lagrangiaanse relaxatie relatief weinig toegepast op machinevolgordeproblemen, geheel ten onrechte. Dit proefschrift laat zien dat voor een scala van machinevolgordeproblemen met behulp van Lagrangiaanse relaxatie zowel betere optimaliserings- als betere benaderingsalgoritmen ontwikkeld kunnen worden.

Hoofdstuk 1 geeft een korte inleiding tot machinevolgordeproblemen, complexiteitstheorie en combinatorische optimalisering; het geeft een uitgebreide inleiding tot Lagrangiaanse relaxatie.

De hoofdstukken 2 tot en met 6 behandelen ieder een specifiek type machinevolgordeprobleem. In Hoofdstuk 2 komen *één-machineproblemen* aan de orde. Lagrangiaanse relaxatie leidt hier tot een *duale* decompositie van dergelijke problemen. Deze decompositie biedt aantrekkelijke mogelijkheden voor de ontwikkeling van optimaliserings- en benaderingsalgoritmen. Rekenexperimenten voor een specifiek één-machineprobleem laten zien dat een benaderingsalgoritme gebaseerd op deze duale decompositie betere resultaten geeft dan een bekend

benaderingsalgoritme.

In Hoofdstuk 3 komen *flow-shop* problemen aan de orde. Lagrangiaanse relaxatie decomponeert het probleem in eerste instantie in verschillende één-machine problemen. Indien men echter een voorwaarde toevoegt die overbodig is voor het oorspronkelijke probleem, dan verkrijgt men een *lineair ordeningsprobleem*. Dit probleem is in algemene zin $\mathcal{NP}$-lastig; voor specifieke waarden van de Lagrangiaanse multiplicatoren kan men het echter in polynomiale tijd oplossen. Het blijkt dat de beste ondergrens die op deze manier verkregen wordt minstens zo goed is als reeds bekende ondergrenzen.

Hoofdstuk 4 behandelt een *parallel-machineprobleem*. Voor dit probleem wordt op basis van Lagrangiaanse relaxatie zowel een benaderings- als een optimaliseringsalgoritme ontwikkeld. De benaderingsalgoritme is een locale zoekmethode waarbij de zoekrichting voorgeschreven wordt door de Lagrangiaanse multiplicatoren. In doorsnee geeft deze methode betere resultaten dan bekende algoritmen voor dit probleem. De optimaliseringsalgoritme kan problemen van behoorlijke omvang in redelijke tijd aan.

In Hoofdstuk 5 komt het *common due date* probleem aan de orde. In dit één-machine probleem hebben alle orders een gemeenschappelijke aflevertijd en worden niet alleen te late maar ook te vroege leveringen bestraft. Een dergelijke visie past in het *just-in-time* principe. Problemen voortvloeiend uit dit principe, en dit probleem in het bijzonder, staan in het middelpunt van de belangstelling. Hoewel het *common due date* probleem in theoretische zin $\mathcal{NP}$-lastig is, blijkt het in praktische zin gemakkelijk te zijn. Met behulp van Lagrangiaanse relaxatie wordt zowel een ondergrens als een bovengrens berekend die bijna altijd aan elkaar gelijk blijken te zijn.

Hoofdstuk 6 behandelt, evenals Hoofdstuk 5, een *just-in-time* probleem met dien verstande dat nu iedere order zijn eigen aflevertijd kent. Hierdoor krijgt men te maken met een specifiek aspect van *just-in-time* problemen: het ongebruikt laten van de machine tussen twee orders in kan voordelig zijn. Lagrangiaanse relaxatie is hier weliswaar nuttig, maar niet zo succesvol als bij andere problemen. Dit *just-in-time* probleem blijkt in rekenkundige zin zeer lastig.

STELLINGEN

behorende bij het proefschrift van

STEVEN LEENDERT VAN DE VELDE

MACHINE SCHEDULING AND LAGRANGIAN RELAXATION

I

Beschouw het volgende probleem. Een verzameling van n orders $\mathcal{J} = \{J_1, \ldots, J_n\}$ dient verwerkt te worden door een enkele machine. Deze machine is beschikbaar vanaf tijdstip 0 en kan niet meer dan één order tegelijkertijd verwerken. Het verwerken van J_j vergt een tijd p_j. De orders hebben een gemeenschappelijke aflevertijd d waarvoor geldt dat $d < \Sigma_{j=1}^{n} p_j$. Zonder verlies van algemeenheid mag men aannemen dat d en de p_j's geheeltallig zijn. Een plan specificeert voor iedere order J_j een completeringstijd C_j zodanig dat aan de beschikbaarheid en de capaciteit van de machine wordt voldaan. Bepaal nu een plan dat de kosten $\Sigma_{j=1}^{n} \mid C_j - d \mid$ minimaliseert. Hall, Kubiak en Sethi [1991] geven een pseudo-polynomiale algoritme die dit probleem oplost in $O(n\Sigma_{j=1}^{n} p_j)$ tijd en ruimte. Men kan het probleem zelfs oplossen in $O(nd)$ tijd en ruimte.

N.G. HALL, W. KUBIAK, S.P. SETHI (1991). Deviation of completion times about a common due date. Te verschijnen in *Operations Research.*

II

Hoewel het *common due date* probleem (zie Stelling I van dit proefschrift) in theoretische zin $\mathcal{NP}$-lastig is, is het in praktische zin gemakkelijk.

J.A. HOOGEVEEN, S.L. VAN DE VELDE (1991). Scheduling around a small common due date. Te verschijnen in *European Journal of Operational Research.*

S.L. VAN DE VELDE (1991). Dit proefschrift, hoofdstuk 5.

III

Beschouw het volgende probleem. Een verzameling van n orders $\mathcal{J} = \{J_1, \ldots, J_n\}$ moet door een enkele machine worden verwerkt. Deze machine begint met de eerste order op tijdstip 0 en kan slechts één order tegelijkertijd verwerken. De orders hebben een gemeenschappelijke kritieke tijd d. De verwerkingstijd van J_j is een functie van het tijdstip t waarop aan J_j begonnen wordt: $p_j(t) = a_j + \max\{0, w_j(t-d)\}$, waarbij a_j de gegeven minimale verwerkingstijd en w_j een gegeven positieve scalair is. Zonder verlies van algemeenheid mag men aannemen dat d, de a_j's en de w_j's geheeltallig zijn. Bepaal nu een volgorde van orders zodanig dat de machine zo spoedig mogelijk klaar is. Kunnathur en Gupta [1990] presenteren een *branch-and-bound* algoritme voor de oplossing van dit probleem. Het probleem kan ook opgelost worden door een pseudo-polynomiale algoritme die $O(nd\Sigma_{j=1}^{n} p_j)$ tijd en $O(nd)$ ruimte vergt.

A.S. KUNNATHUR, S.K. GUPTA (1990). Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research 47*, 56-64.

IV

Townsend [1978] en Gupta en Sen [1983] gebruiken de zogeheten *maximum potential improvement method* om een ondergrens te berekenen voor de minimale waarde van een kwadratische functie van de completeringstijden van orders op één machine. De ondergrens wordt verkregen door een specifieke bovengrens voor de minimale waarde te verminderen met de af te schatten *maximum potential improvement*. Deze afschatting is onnodig zwak.

S.K. GUPTA, T. SEN (1983). Minimizing a quadratic function of job lateness on a single machine. *Engineering Costs and Production Economics 7*, 187-194.

W. TOWNSEND (1978). The single machine problem with quadratic penalty function of completion times: a branch-and-bound solution. *Management Science 24*, 530-534.

V

De *maximum potential improvement method* (zie Stelling IV van dit proefschrift) is ook toegepast op problemen met samengestelde functies van de completeringstijden; zie o.a. Sen, Raiszadeh en Dileepan [1988]. *Objective splitting* domineert deze methode en is bovendien eenvoudiger.

J.A. HOOGEVEEN, S.L. VAN DE VELDE (1990). *A new lower bound approach for single-machine multicriteria scheduling*, Report BS-R9026, CWI, Amsterdam.

T. SEN, F.M.E. RAISZADEH, P. DILEEPAN (1988). A branch-and-bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness. *Management Science 34*, 254-260.

VI

De tweede ondergrens gepresenteerd door Bozoki en Richard [1970] is fout.

G. BOZOKI, J.-P. RICHARD (1970). A branch-and-bound algorithm for the continuous-process job-shop scheduling problem. *AIIE Transactions 2*, 246-252.

VII

Dirickx, Baas en Dorhout [1987] stellen dat Lagrangiaanse relaxatie zinloos is in geval het Lagrangiaanse probleem de geheeltalligheidseigenschap bezit. Deze bewering gaat voorbij aan de afweging tussen snelheid en kwaliteit.

Y.M.I. DIRICKX, S.M. BAAS, B. DORHOUT (1987). *Operationele research*, Academic Service, Schoonhoven.

VIII

Het verdient aanbeveling de geplande hogesnelheidstrajecten parallel aan snelwegen aan te leggen; een snellere trein demoraliseert de automobilist.

IX

De virtuele prijs die een beursstudent voor de OV-jaarkaart betaalt rechtvaardigt het herinvoeren van de derde klasse in het openbaar vervoer.

X

In de Algemene Richtlijnen bij Promoties van de Technische Universiteit Eindhoven staat het volgende. 'Als laatste stelling wordt het de promovendus gegund om zijn wijsheid te laten schijnen op onderwerpen van zeer uiteenlopende aard. Hierbij is het gewenst dat de inhoud en/of de vorm een zekere verrassende, soms paradoxale, zelfs enigszins provocerende inhoud heeft. Zo'n stelling wordt soms aangeduid als schertsstelling.' Dit is een schertsrichtlijn.