Special Flowshop Problems to Minimize Total Completion Time

By Masanori Okada

A thesis submitted to
School of Information System Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Doctor of Information System Science
Graduate Program in Information Science

Written under the direction of
Professor Milan Vlach

January 16, 1998

Abstract

In this dissertation we are concerned with deterministic flowshop problems where the objective is to minimize the sum of completion times of all jobs. With a few exceptions flowshop problems of this type are known to be computationally intractable. Therefore we have restricted our attention to the following two special cases.

The first case deals with two types of specially structured dominance among the machines. It is known [1, 7] that under such a dominance the best schedule among the so called "permutation schedules" can be found in polynomial time. Here we prove that the schedules constructed as in [1, 7] are not only the best permutation schedules but are the optimal ones. In fact we prove a much more general result, namely that under the above machine dominance constraints, the search for optimal schedule can be restricted to the set of all permutation schedules not only for the sum of completion times criterion but for an arbitrary regular objective function.

The second case deals with two machine problems. We study the two machine problems under the assumption that no idle machine time between the consecutive operation is allowed. First we show that some claims in the literature are incorrect. Then we prove statements which have similar "flavor" as the original incorrect claims. From Chapter 5 on, the focus of our study is on the subproblem specified by the additional constraint that all operations on the first machine have the same length. We describe several variants of the branch and cut technique for solving this strongly NP-hard problem and report results of computational experiments.

Acknowledgments

The author wishes to express his sincere gratitude to his principal advisor Professor Milan Vlach of Japan Advenced Institute of Science and Techology for his constant encouragement and kind guidance during this work. The author would like to thank his advisor Associate Professor Kunihiko Hiraishi of Japan Advanced Institute of Science and Technology.

The author is grateful to Professor Hiroaki Ishii of Osaka University for their helpful suggestions and discussions.

The author also wishes to express his thanks to Professor Masayuki Kimura and Tetuso Asano of Japan Advanced Institute of Science and Technology for their suggestions.

The author is grateful to all who have affected or suggested his areas of research. Visiting Associate Ondřej Čepek of Japan Advanced Institute of Science and Technology inspired the author through his constant activities on scheduling theory, and gave the author valuable suggestions and kind encouragements.

The author devotes his sincere thanks and appreciation to all of them, and his colleagues.

Contents

Chapter 1

Introduction

Sequencing and scheduling theory is concerned primarily with the development of mathematical models and techniques for analyzing and solving problems arising in situations in which scarce resources have to be allocated to activities over time. The early research was motivated mainly by problems from manufacturing and processing industries. The resources were usually machines and activities were jobs to be executed on the machines. However, analogous problems arise in such a variety of situations that the number of problem types is practically unlimited. For example, several new classes of practically relevant and theoretically interesting scheduling problems have come to existence in connection with the development of flexible manufacturing systems and technological advances in robotics. Another fundamentally new direction has been connected with a rapid expansion of computer science. This influence has taken several forms. On the one hand, results in complexity theory led to deeper understanding of the inherent difficulty of scheduling problems and to devising new techniques for obtaining bounds on performance of approximation algorithms. On the other hand, scheduling theory has devised new models in studies of multiprocessors systems, distributed systems and computer networks. The authors in this area refer to machines as processors and to jobs as tasks. We shall be using the classical terminology, that is we shall be talking about jobs and machines.

In this dissertation we are concerned only with cases where all problem data are exact and known in advance. These deterministic problems have the form of the following optimization problem: Given a finite number of jobs $J_1, J_2, \cdots, J_n$ which have to be processed on a system of a finite number of machines $M_1, M_2, \cdots, M_m$, find a feasible schedule that minimize the value of a given objective function.

Before specifying exactly the special structured class of deterministic problems which the dissertation is devoted to, we shall briefly recall some basic models of deterministic machine scheduling. In all these models the following three assumptions are conventional.

- Each machine is always available throughout a given scheduling period, usually the interval $[0, \infty)$.
- No machine is able to process more than one job at a time.
- Each job can be processed by at most one machine at a time.

It is an established tradition to distinguish three types of multi-operation models: openshops, jobshops and flowshops. In all these three models, each jobs J_k consists of m operations $O_{1k}, O_{2k}, \cdots, O_{mk}$. Each operation O_{ik} must be executed on machine M_i and

has a positive processing time p_{ik}. In the openshop, no particular processing order through the machines is prescribed. In the jobshop and flowshop a processing order through the machines is prescribed for each job. In the jobshop this order may vary from job to job, whereas in the flowshop all jobs have the same order through the machines. If $m = 1$, then all three models reduce to the single machine problem.

Generally speaking, a schedule must specify when the jobs are processed on the machines within the scheduling period. The assumptions stated above make it possible to express the required information with the help of an ordered m-tuple $S = (S_1, S_2, \cdots, S_m)$ of functions each of which maps the scheduling period into the set $\{0, 1, \cdots, n\}$. The corresponding interpretation is rather obvious. Namely, the equality $S_i(t) = k$ means that jobs J_k is processed on machine M_i at time t, whenever $k \neq 0$. If $k = 0$, that is if $S_i(t) = 0$, then machine M_i is idle at time t.

Of course not every ordered m-tuple $(S_1, S_2, \cdots, S_m)$ of such piece-wise constant functions represents a feasible schedule. Besides some purely mathematical technicalities we have to require that the assumptions above are satisfied and that each job must be processed to completion within the scheduling period. In addition to these basic feasibility requirements, several other constraints may limit the choice of schedules. In most cases they represent limits on the capacity of available resources or various types of technological or organizational requirements. For example, the feasibility may depend on whether or not

- the processing of a job on a machine may be interrupted and resumed at a later time;
- a precedence ordering is imposed on the jobs;
- job-dependent deadlines or release times are given.

All such constraints can be formulated in terms of functions $S_1, S_2, \cdots, S_m$. In this way we obtain a well-defined concept of a feasible schedule which can be easily graphically represented, manipulated and analyzed.

As soon as it is clear which schedules are feasible, then the first question to be asked is whether a feasible schedule exists. In general, this question may be very hard, because already the problem of deciding whether or not a feasible schedule exists is NP-complete in the single-machine case, provided arbitrary integer processing times, release dates and deadlines are permitted. However, in the dissertation, we are concerned with problems in which this question is trivial. In fact, we shall deal with situations in which there exist infinite number of feasible schedules. In such situation we need a tool for their mutual comparison. The standard approach is to compare the quality of schedules with the help of a real valued function f defined on the set of schedules in such a way that schedule S is better than schedule S' if and only if $f(S) < f(S')$. Such function is called an objective function, and the problem is to minimize f on the set of feasible schedules. Usually the objective functions of the basic models are so called regular measures of performance. By this term it is meant that f depends on the completion times $C_1(S), C_2(S), \cdots, C_n(S)$ of jobs in schedule S in such a way that if $C_j(S) \leq C_j(S')$ for each $1 \leq j \leq n$, then $f(S) \leq f(S')$. In other words, if schedule S is better than S', then at least one job must be completed under schedule S earlier than under schedule S'.

Essentially two types of regular objective functions are appearing in the basic models. They are composed from nondecreasing real valued functions $\varphi_1, \varphi_2, \cdots, \varphi_n$ associated

with individual jobs as follows:

$$\begin{aligned} f_{max}(S) &= max_{1\leq k\leq n}\varphi_k(C_k(S)), \\ f_{sum}(S) &= \sum_{i=1}^{n}\varphi_k(C_k(S)). \end{aligned}$$

As a rule, these so called cost or penalty functions $\varphi_1, \varphi_2, \cdots, \varphi_n$ are based on extremely simple quantities involving explicitly or implicitly the concept of job due dates. As typical example we can mention

$$\begin{aligned} \text{Lateness} &: \varphi_k(t) = L_k(t) = t - d_k \\ \text{Tardiness} &: \varphi_k(t) = T_k(t) = \max\{0, t - d_k\} \\ \text{Unit Penalty} &: \varphi_k(t) = U_k(t) = \begin{cases} 1 & \text{if } t > d_k \\ 0 & \text{if } t \leq d_k \end{cases} \end{aligned}$$

where d_k stands for the due date of jobs J_k.

Several classification and notation scheme for deterministic scheduling problems have been suggested. We have followed the scheme proposed by Graham et al. [6] in which a problem type is specified in term of the following three-field classification $\alpha|\beta|\gamma$. The first field specifies the machine environment. For example, if $\alpha = J$ then we have general jobshop, and if $\alpha = F2$, then we have the two machine flowshop problem. the second field is concerned with job characteristics. For example, if $\beta = prec$ then a general precedence relation is permitted, and if $\beta = pmtn$ then preemption is allowed. The third field refers to the objective function. For example: if $\gamma = T_{max}$, then the maximum tardiness is to be minimized, and if $\gamma = \sum U_j$ then the number of tardy jobs should be minimized. For a detailed description of the scheme and a survey of complexity and algorithms we refer to [12].

In this dissertation we are concerned with a special class of deterministic flowshop problems. As already mentioned, in the deterministic flowshop problems, a set of n jobs $J_1, J_2, \cdots, J_n$ is to be processed through m machines $M_1, M_2, \cdots, M_m$ under the technological constraints demanding that the jobs pass among the machines in the same order. Without loss of generality we may assume that the machines are numbered according to the technological constraints, that is we may assume that each job must be processed first on machine M_1, then on machine M_2, and so on, until it is finally processed on machine M_m. The order of jobs on any given machine is not prescribed and may vary from machine to machine. No machine may process more than one job at a time, and no job can be processed by several machines simultaneously. Each job consists of m operations, one operation per machine. The operation of job J_j on machine M_i is denoted by O_{ij} and the processing time of operation O_{ij} is denoted by p_{ij}. Once a processing of an operation O_{ij} starts, it cannot be interrupted, i.e. operation O_{ij} then occupies the machine M_i for the next p_{ij} time units. In scheduling theory literature operations fulfilling the last condition are called nonpreemptable. Some other technological or organizational constraints on the feasibility of schedules may also be given. The general problem is then to find a feasible schedule minimizing a prescribed objective regular function defined on the set of feasible schedules.

A great variety of such problems have been studied by both theorists and practitioners since the pioneering work of J. M. Johnson [10] from mid 50's, which dealt with two-machine and three-machine problems. In this dissertation we consider flowshop problems

where the objective is to minimize the sum of completion times of all jobs. Since this task is known to be very hard in general, we shall restrict our attention to two subcases. The first subcase studied in Section 3 deals with two types of specially structured dominance among the machines. It is known [1, 7] that under such a dominance the best schedule among the so called "permutation schedules" can be found in polynomial time. Here we prove that the schedules constructed as in [1, 7] are not only the best permutation schedules but are the best among all feasible schedules, i.e. are optimal. In fact we prove a much more general result, namely that under the above machine dominance constraints, the search for optimal schedule can be restricted to the set of all permutation schedules not only for the sum of completion times criterion but for an arbitrary regular objective function which fulfills certain very general properties.

The second subcase which we study in Section 4 is the case of two machines. Following the notation and terminology of the classification for the deterministic scheduling problems proposed by Graham et al. [6], we denote this problem as $F2||\sum C_j$. One of the first papers to study this particular problem was [9] where a branch-and-bound algorithm for the problem was developed. The complexity of the problem was later established in [5]. It was proved that $F2||\sum C_j$ is NP-hard in the strong sense. Given this intractability result, it comes as a no surprise that a lot of effort was put into developing heuristics and approximation algorithms based on several lower bound techniques (see e.g. [11, 15] or [4] for survey). All the above algorithms utilize the following fact which is a consequence of a more general theorem in [3]: it suffices to optimize over a set of those schedules in which both machines process jobs in the same order and moreover in which the first machine has no idle time.

This raises a question how does the complexity of $F2||\sum C_j$ change if we allow no idle time also on the second machine. We denote this new problem as $F2|nmit|\sum C_j$ where "*nmit*" stands for "no machine idle time". Such a restriction may be very natural in some real life situation, e.g if both machines represent an expensive equipment which has to be rented for the duration between the start of the first operation and the completion of the last one. First thing to note is that the argument from [3] carries over for this no-idle case, i.e. again it suffices to optimize over the set of permutation schedules. To make this dissertation self-contained, we recall in Section 4 the proof of this fact. We then commence the study of $F2|nmit|\sum C_j$ by stating two theorems from [1] which deal with certain properties of optimal schedules. We show that both claims are incorrect. Moreover, we manage to prove another statement which has a similar "flavor" as the original (incorrect) claims. Similarly to the argument from [3], also the complexity result from [5] carries over to $F2|nmit|\sum C_j$. Therefore it is legitimate to decompose the problem yet further, and study subproblems of $F2|nmit|\sum C_j$.

The subproblem that will be in the focus of our study from Section 5 on has the following additional constraint: all operations on the first machine have a uniform length. The complexity of this $F2|p_{1j}=a,nmit|\sum C_j$ problem was recently investigated in [8]. Once again, it was proved that the problem is NP-hard in the strong sense. Although strictly speaking the authors of [8] work with $F2|p_{1j}=a|\sum C_j$, i.e. they do not require the "no-idle" constraint, their proof is valid without any changes for the $F2|p_{1j}=a,nmit|\sum C_j$ case as well. In Section 5 we describe several variants of the branch-and-cut technique for solving the $F2|p_{1j}=a,nmit|\sum C_j$ problem and report results of computational experiments.

Chapter 2

Basic definitions, notation, and results

Let us start this chapter by introducing some fairly standard notions and stating several simple results. First we need to formalize the notion of a schedule. A *schedule* S for a flowshop problem with n jobs on m machines is a set of mn nonnegative numbers C_{ij}^S, $1 \leq i \leq m$, $1 \leq j \leq n$, where each C_{ij}^S denotes the completion time of the operation O_{ij} in the schedule S. Since the operations cannot be preempted, a system of completion times C_{ij}^S fully specifies a schedule as described in the Introduction. In other words, since the operations can not be preempted the completion time fully specifies the span in which each operation is processed, i.e. operation O_{ij} is processed in the interval $(C_{ij}^S - p_{ij}, C_{ij}^S)$, where p_{ij} denotes the prescribed duration of operation O_{ij}. For every job J_j, $1 \leq j \leq n$, the job completion time C_j^S is the completion time of its last operation, i.e. $C_j^S = C_{mj}^S$.

A schedule S is called *feasible* if it fulfills all *feasibility constraints*. In addition to the obvious requirement $C_{1j}^S - p_{1j} \geq 0$, there are two feasibility constraints for general flowshop problems:

- Machine constraint: each machine may process at most one job at any given time. Formally

$$\forall i \in \{1, \ldots, m\} \forall k, \ell \in \{1, \ldots, n\} : (k \neq \ell) \Longrightarrow (C_{ik}^S - p_{ik}, C_{ik}^S) \cap (C_{i\ell}^S - p_{i\ell}, C_{i\ell}^S) = \emptyset.$$

- Job constraint: no job is processed on more than one machine simultaneously, and moreover the operations of each job are processed on all machines in the same order. Formally

$$\forall i, j \in \{1, \ldots, m\} \forall k \in \{1, \ldots, n\} : (i < j) \Longrightarrow (C_{ik}^S \leq C_{jk}^S - p_{jk}).$$

In the main part of this dissertation we will require the schedules to fulfill an additional feasibility constraint:

- No-idle constraint: no machine is allowed to have an idle time between processing any two operations. Formally

$$\forall i \in \{1, \ldots, m\} \forall k, \ell \in \{1, \ldots, n\} : (C_{ik}^S < C_{i\ell}^S - p_{i\ell}) \Longrightarrow (\exists q : C_{ik}^S < C_{iq}^S < C_{i\ell}^S).$$

The quality of a schedule is measured by an *objective function* which assigns to every schedule a real number. The task is then to find a feasible schedule which attains the minimum value of the objective function over the set of all feasible schedules. Every schedule with this property is called *optimal*. Typically, the objective function depends only on the job completion times and furthermore possesses a regularity property. An objective function f is said to be *regular* if for every two distinct schedules S_1 and S_2 the inequality $f(S_1) < f(S_2)$ implies $C_j^{S_1} < C_j^{S_2}$ for at least one j. The objective function studied in this dissertation is $f(S) = \sum_{j=1}^{n} C_j^S$. Clearly, this is a regular objective function.

Let us observe that because there are no deadlines on the completion times of jobs, there exists a feasible schedule for every instance of the above described no-idle flowshop problem (which is then of course also a feasible schedule for the "idle time allowed" version of the problem). Given an instance of the problem, such a schedule can for instance be created as follows: schedule consecutively all operations on the first machine (in an arbitrary order with no idle time allowed) starting from time 0, then schedule consecutively all operations on the second machine (in an arbitrary order with no idle time allowed) starting from time $\sum_{j=1}^{n} p_{1j}$ (i.e. starting when the last operation on the first machine is completed) and so on. It is easy to verify that such a schedule is indeed feasible. Moreover, since there are no deadlines on the completion times of jobs, any feasible schedule S can be "shifted right" by an arbitrary real number r (replacing every C_{ij}^S by $C_{ij}^S + r$) resulting again in a feasible schedule. Therefore there is an uncountable number of feasible schedules for every instance of every of the flowshop problems being considered. In order to reduce the number of schedules under investigation, we need the notion of dominance.

A set of schedules $\mathcal{S}$ is said to be *dominant* if for every feasible schedule S there exists a feasible schedule $S' \in \mathcal{S}$ such that $f(S') \leq f(S)$. This guarantees that at least one optimal schedule lies in $\mathcal{S}$ and hence it is enough to optimize over $\mathcal{S}$ and disregard all schedules not in $\mathcal{S}$. Now let us show that as long as the objective function is regular there exists a finite size dominant set of schedules.

Due to the non-preemptiveness requirement every schedule defines for each machine an order in which the jobs are processed, or in other words a permutation of jobs. Of course the same set of m permutations (one per each machine) may be defined by many different schedules. However, it is not hard to see that for each fixed set of m permutations, say $\pi_1, \ldots, \pi_m$, there is one schedule which dominates all the others in the same group. This schedule (let us denote it by $S_{\pi_1,\ldots,\pi_m}$) can be constructed as follows:

1. Schedule all operations on the first machine in the order π_1 with no idle time in between operations starting at time 0.

2. Schedule all operations on the second machine in the order π_2 as follows:

   - If idle time is allowed then schedule the operations one by one in the order π_2 where each operation starts as early as possible, i.e. starts at the time when the operation of the same job on the previous machine is completed or at the time when the previous operation on the same machine is completed, whichever comes last. More formally, the completion time of an operation $O_{2\pi_2(i)}$ for each $1 \leq i \leq n$ is defined by

     $$C_{2\pi_2(i)}^S = \max\{C_{1\pi_2(i)}^S, C_{2\pi_2(i-1)}^S\} + p_{2\pi_2(i)},$$

     where we assume $C_{2\pi_2(0)}^S = 0$ and $S = S_{\pi_1,\ldots,\pi_m}$ to simplify the notation.

- If idle time is not allowed then schedule all the operations in the given order with no idle time in between the operations starting at the time when the last operation on the previous machine was completed. Then "shift" all the operations to the left until the starting time of one of the operations "hits" the completion time of its corresponding operation on the previous machine. The job that prohibits further shifting to the left (in case that there are several such jobs then the leftmost one) is called the blocking job of S for the second machine and its position in S is called the blocking position of S for the second machine (see Figure 2.1).

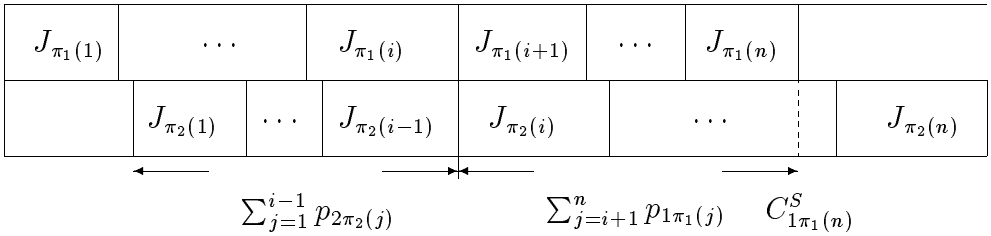

Figure 2.1:

More formally, the operation $O_{2\pi_2(i)}$ (for each $1 \leq i \leq n$) is first assigned a "tentative completion time" $\hat{C}^S_{2\pi_2(i)} = C^S_{1\pi_1(n)} + \sum_{j=1}^{i} p_{2\pi_2(j)}$ and then the (final) completion time is calculated by subtracting the "length of the left shift" $\ell = \min_{i=1}^{n}\{\sum_{j=1}^{i-1} p_{2\pi_2(j)} + \sum_{j=i+1}^{n} p_{1\pi_1(j)}\}$. Hence

$$C^S_{2\pi_2(i)} = \hat{C}^S_{2\pi_2(i)} - \ell = C^S_{1\pi_1(n)} + \sum_{j=1}^{i} p_{2\pi_2(j)} - \min_{i=1}^{n}\{\sum_{j=1}^{i-1} p_{2\pi_2(j)} + \sum_{j=i+1}^{n} p_{1\pi_1(j)}\}.$$

3. Repeat step 2 for machines $M_3, M_4, \ldots, M_m$.

By construction it is obvious that no operation in $S_{\pi_1,\ldots,\pi_m}$ can be shifted left without violating feasibility. On the other hand given any feasible schedule S which defines the same set of permutations $\pi_1, \ldots, \pi_m$, it is possible to transform S into $S_{\pi_1,\ldots,\pi_m}$ by performing left shifts on a certain set of operations.[1] Therefore $f(S_{\pi_1,\ldots,\pi_m}) \leq f(S)$ holds for every regular objective function f. Consequently the set $\mathcal{S} = \{S_{\pi_1,\ldots,\pi_m} \mid \pi_1, \ldots, \pi_m \in P_n\}$ where P_n is the set of all permutations of order n is a dominant set of schedules. Note that $|\mathcal{S}| = (n!)^m$, i.e. $\mathcal{S}$ has a finite size. From now on we shall restrict our attention exclusively to schedules from the set $\mathcal{S}$.

For some flowshop problems it is possible to restrict the set $\mathcal{S}$ even further to a set of the so-called permutation schedules. A schedule $S_{\pi_1,\ldots,\pi_m} \in \mathcal{S}$ is a *permutation schedule* if $\pi_1 = \pi_2 = \cdots = \pi_m$ holds, i.e. if the jobs are processed in the same order on all m machines. The name "permutation schedule" reflects the fact that a single permutation specifies the entire schedule. We denote the set of all permutation schedules by $\mathbf{S}$.

Throughout this dissertation we shall make an assumption that all processing times p_{ij} are integers. It then follows from the construction of the schedules in $\mathcal{S}$ that also all completion times of operations can be assumed integral.

[1] We omit a formal proof of this fact here. However, the reader can easily verify the claim by induction, starting with the leftmost operation on which the schedules S and $S_{\pi_1,\ldots,\pi_m}$ differ.

Chapter 3

Flowshop with machine dominance

As we already mentioned in the introduction, the problem $Fm||\sum C_j$ is NP-hard for $m \geq 2$ [5]. That implies that there is little hope to find a polynomial time algorithm which would generate an optimal solution for this problem in its full generality. Hence, a question arises what additional constraints should be imposed on the problem in order to make it tractable. Several such constraints are connected to the long-studied concept of machine dominance [1, 14, 7]. A machine i is said to *dominate* a machine j (denoted by $M_i \succ M_j$) if

$$\min_{k=1}^{n}\{p_{ik}\} \geq \max_{k=1}^{n}\{p_{jk}\}.$$

We shall deal here with two special cases studied in [1] and [7]. The machines $M_1, \ldots, M_m$ are said to form an *increasing series of dominating machines (idm)* if

$$M_m \succ M_{m-1} \succ \cdots \succ M_2 \succ M_1,$$

and to form a *decreasing series of dominating machines (ddm)* if

$$M_1 \succ M_2 \succ \cdots \succ M_{m-1} \succ M_m.$$

Both cases are known to be solvable in polynomial time if we restrict our attention to permutation schedules only. Adiri and Pohoryles [1] designed two polynomial time algorithms constructing the best permutation schedules for $Fm|nmit, idm|\sum C_j$ and $Fm|nmit, ddm|\sum C_j$ respectively. Ho and Gupta [7] later extended these results for the case when machine idle time is allowed. In this dissertation we prove that all of the above algorithms construct not only the best permutation schedule for the given problem, but that they in fact produce optimal schedules. This is achieved by proving that under either of the above machine dominance schemes (*idm* or *ddm*) the set of all permutation schedules **S** is a dominant set of schedules. Moreover, our results are more general in the sense, that they are valid not only for the $\sum C_j$ criterion, but rather for an arbitrary regular objective function. Let us start with the *idm* case.

Proposition 1 *Let f be an arbitrary regular objective function. Then the set* **S** *of all permutation schedules is a dominant set for both $Fm|idm|f$ and $Fm|nmit, idm|f$ flowshop problems.*

Proof. First let us note that if the machines form an increasing series of dominating machines then it is not necessary to distinguish the situations when machine idle time

is allowed and when it is not allowed. For this it is enough to show that under the *idm* constraint no schedule $S \in \mathcal{S}$ may have any idle time inbetween any two consecutively scheduled operations. However, this is a straightforward consequence of the *idm* constraint and the method how the schedules in $\mathcal{S}$ are constructed. Also note that for every schedule $S \in \mathcal{S}$, the job scheduled as the first one on M_i is the blocking job for M_i, $1 \leq i \leq m$.

Let $S \in \mathcal{S}$ be arbitrary. We shall construct $T \in \mathbf{S}$ such that $f(T) \leq f(S)$. For that let π be the permutation of jobs defined by the schedule S on the last machine, and let us denote $j = \pi(1)$ (job j is scheduled first on M_m in S). Before constructing T we first define a schedule R by modifying S as follows:

- On all machines schedule job j at the same time as in S, i.e. set $C^R_{ij} = C^S_{ij}$ for all $1 \leq i \leq m$.
- On all machines schedule the remaining jobs in an order given by the permutation π with no idle time inbetween operations. Since the job j is already scheduled, it determines the completion times of all remaining operations on all machines. Formally: $C^R_{i\pi(k)} = C^R_{ij} + \sum_{\ell=2}^{k} p_{i\pi(\ell)}$ for all $1 \leq i \leq m$ and $2 \leq k \leq n$.

First of all let us note that $f(R) = f(S)$ because the schedule on the last machine is identical in R and S. Secondly let us prove that R is a feasible schedule. By contradiction let us assume that R is infeasible. Obviously, the only feasibility constraint that R may violate is the job constraint. So let the job $\pi(k)$ be such that $O_{i_1\pi(k)}$ is completed only after $O_{i_2\pi(k)}$ already starts for some machines $i_1 < i_2$, i.e. let $C^R_{i_1\pi(k)} > C^R_{i_2\pi(k)} - p_{i_2\pi(k)}$. After substituting we get

$$C^R_{i_1 j} + \sum_{\ell=2}^{k} p_{i_1\pi(\ell)} > C^R_{i_2 j} + \sum_{\ell=2}^{k} p_{i_2\pi(\ell)} - p_{i_2\pi(k)} = C^R_{i_2 j} + \sum_{\ell=2}^{k-1} p_{i_2\pi(\ell)}.$$

Moreover, since S is a feasible schedule we also have

$$C^R_{i_2 j} - p_{i_2 j} = C^S_{i_2 j} - p_{i_2 j} \geq C^S_{i_1 j} = C^R_{i_1 j}.$$

Putting the above two inequalities together we get

$$C^R_{i_1 j} + \sum_{\ell=2}^{k} p_{i_1\pi(\ell)} > C^R_{i_1 j} + p_{i_2 j} + \sum_{\ell=2}^{k-1} p_{i_2\pi(\ell)} = C^R_{i_1 j} + \sum_{\ell=1}^{k-1} p_{i_2\pi(\ell)}$$

which implies

$$\sum_{\ell=2}^{k} p_{i_1\pi(\ell)} > \sum_{\ell=1}^{k-1} p_{i_2\pi(\ell)}$$

which is a contradiction to the fact that $M_{i_2} \gg M_{i_1}$. Therefore the schedule R is feasible. Now we are ready to construct from the schedule R the schedule $T \in \mathbf{S}$. This is done by simply eliminating on each machine the (maybe zero length) unnecessary idle time before the first job starts. This is equivalent to shifting the schedule on each of the machines $M_2, \ldots, M_m$ to the left until the job j becomes the blocking job. This implies that all job completion times can only decrease (i.e. $C^T_k \leq C^R_k$ for every $1 \leq k \leq n$) which together with the fact that f is regular guarantees that $f(T) \leq f(R) = f(S)$. ■

Let us now turn to the *ddm* case. Unlike in the *idm* case, this time it is necessary to distinguish whether a machine idle time is allowed or not. Let us demonstrate this fact on an example in which we take $f = \sum C_j$.

Example 1 Consider the 3-job instance with $p_{11} = p_{12} = 3$, $p_{13} = 4$, $p_{21} = p_{22} = 1$, and $p_{23} = 2$. According to [7] the best permutation schedule for $F2|ddm|\sum C_j$ is the one with an SPT order (jobs sorted by length from short to long) on the first machine, for instance the schedule corresponding to the permutation $\langle 1, 2, 3\rangle$ (see Figure 3.1). On the other hand, according to [1] the best permutation schedule for $F2|nmit, ddm|\sum C_j$ is the one with an SPT order on the second machine, except maybe the last job. Since $p_{11} = p_{12}$ and $p_{21} = p_{22}$ it suffices to check the schedules where J_3 is last on M_2 and where (say) J_2 is last on M_2, for instance the schedules corresponding to permutations $\langle 1, 2, 3\rangle$ and $\langle 1, 3, 2\rangle$. It can be easily verified that the latter is the better of the two (see Figure 3.1). Note that not only may two schedules corresponding to the same permutation under the

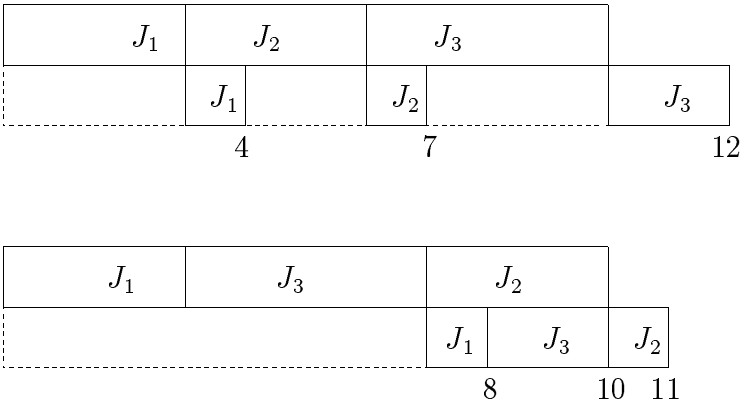

Figure 3.1:

no-idle constraint and without this constraint be different (which implies that the sets $\mathcal{S}$ and also $\mathbf{S}$ are different), but moreover that the best permutation schedule is in each case attained for a different permutation of jobs. This shows that the problems $Fm|ddm|\sum C_j$ and $Fm|nmit, ddm|\sum C_j$ are genuinely different even for $m = 2$.

Now we are ready to state the variants of Proposition 1 for the problems $Fm|ddm|f$ and $Fm|nmit, ddm|f$.

Proposition 2 *Let f be an arbitrary regular objective function. Then the set $\mathbf{S}$ of all permutation schedules is a dominant set for the $Fm|ddm|f$ flowshop problem.*

Proof. Let $S \in \mathcal{S}$ be arbitrary. We shall construct $T \in \mathbf{S}$ such that $f(T) \leq f(S)$. Let us assume that the jobs are numbered according to their order on M_1 in S. Let M_i be the first machine which has in S a different order of jobs than M_1, and let j be the first (leftmost) job scheduled on M_i out of the sequence defined by M_1. Let k be the job that precedes j on M_i in S. Let us observe several simple facts:

- Since the order of jobs $J_1, \ldots, J_k$ is the same on M_{i-1} and M_i in S, the operation O_{ik} starts immediately after the conclusion of the operation $O_{(i-1)k}$. This is a simple consequence of the *ddm* constraint. Moreover, because $p_{(i-1)(k+1)} \geq p_{ik}$ we have $C^S_{(i-1)(k+1)} \geq C^S_{ik}$.

- Since $j \geq k+2$ we also get $C^S_{(i-1)(k+2)} \leq C^S_{ij} - p_{ij}$.

The above two facts imply that M_i has an idle time of length at least $p_{(i-1)(k+2)} \geq p_{i(k+1)}$ between $C^S_{(i-1)(k+1)}$ and the start of operation O_{ij}. However, that means that the operation $O_{i(k+1)}$ which is scheduled after O_{ij} in S can be moved inbetween O_{ik} and O_{ij} to start at $C^S_{(i-1)(k+1)}$. Therefore, after a finite number of such actions, the order of jobs on M_i can be made to coincide with the order on M_1. The same can be subsequently done for all remaining machines producing the schedule T. Since all operations either stayed in their place or were moved left, we have $C^T_j \leq C^S_j$ for all $1 \leq j \leq n$. This together with the fact that f is regular implies $f(T) \leq f(S)$. ■

Proposition 3 *Let f be an arbitrary regular objective function. Then the set $\mathbf{S}$ of all permutation schedules is a dominant set for the $Fm|nmit, ddm|f$ flowshop problem.*

Proof. Let $S \in \mathcal{S}$ be arbitrary. We shall construct $T \in \mathbf{S}$ such that $f(T) \leq f(S)$. For that let σ be the permutation of jobs defined by the schedule S on the last machine, and let $J_j = J_{\sigma(q)}$ be the job scheduled last on M_1 in S. Before constructing T we first define a schedule R by modifying S as follows:

- On all machines schedule job j at the same time as in S, i.e. set $C^R_{ij} = C^S_{ij}$ for all $1 \leq i \leq m$.
- Construct a permutation π from the permutation σ as follows

$$\pi(i) = \begin{cases} \sigma(i) & \text{for } 1 \leq i < q, \\ \sigma(i+1) & \text{for } q \leq i < n, \\ \sigma(q) & \text{for } i = n. \end{cases}$$

  This corresponds to taking the jobs scheduled after J_j on M_m in S and moving them (without changing their order) just in front of J_j.

- On all machines schedule the remaining jobs in an order given by the permutation π with no idle time inbetween operations. Since the job j is already scheduled, it determines the completion times of all remaining operations on all machines. Note that since the job $J_j = J_{\sigma(q)} = J_{\pi(n)}$ is last on M_1 in S, the change to the schedule R amounts on M_1 to reordering the first $n-1$ jobs according to the permutation π, which obviously keeps the starting time of the first scheduled job at time zero. Formally: $C^R_{i\pi(k)} = C^R_{ij} - \sum_{\ell=k+1}^{n} p_{i\pi(\ell)}$ for all $1 \leq i \leq m$ and $1 \leq k \leq n-1$.

First of all let us note that $f(R) \leq f(S)$ because f is regular and the schedule on the last machine in R was obtained from S by shifting some (possibly zero) operations to the left. Secondly let us prove that R is a feasible schedule. By contradiction let us assume that R is infeasible. Obviously, the only feasibility constraint that R may violate is the job constraint. So let the job $\pi(k)$ be such that $O_{i_1\pi(k)}$ is completed only after $O_{i_2\pi(k)}$ already starts for some machines $i_1 < i_2$, i.e. let $C^R_{i_1\pi(k)} > C^R_{i_2\pi(k)} - p_{i_2\pi(k)}$. After substituting we get

$$C^R_{i_1 j} - \sum_{\ell=k+1}^{n} p_{i_1\pi(\ell)} > C^R_{i_2 j} - \sum_{\ell=k+1}^{n} p_{i_2\pi(\ell)} - p_{i_2\pi(k)} = C^R_{i_2 j} - \sum_{\ell=k}^{n} p_{i_2\pi(\ell)}.$$

Moreover, since S is a feasible schedule we also have

$$C^R_{i_2 j} - p_{i_2 j} = C^S_{i_2 j} - p_{i_2 j} \geq C^S_{i_1 j} = C^R_{i_1 j}.$$

Putting the above two inequalities together we get

$$C^R_{i_1 j} - \sum_{\ell=k+1}^{n} p_{i_1 \pi(\ell)} > C^R_{i_1 j} + p_{i_2 j} - \sum_{\ell=k}^{n} p_{i_2 \pi(\ell)} = C^R_{i_1 j} - \sum_{\ell=k}^{n-1} p_{i_2 \pi(\ell)}$$

which implies

$$\sum_{\ell=k+1}^{n} p_{i_1 \pi(\ell)} < \sum_{\ell=k}^{n-1} p_{i_2 \pi(\ell)}$$

which is a contradiction to the fact that $M_{i_1} \gtrdot M_{i_2}$. Therefore the schedule R is feasible. Now we are ready to construct from the schedule R the schedule $T \in \mathbf{S}$. This is done by simply eliminating on each machine the (maybe zero length) unnecessary idle time before the first job starts. This is equivalent to shifting the schedule on each of the machines $M_2, \ldots, M_m$ to the left until the job j becomes the blocking job. This implies that all job completion times can only decrease (i.e. $C^T_k \leq C^R_k$ for every $1 \leq k \leq n$) which together with the fact that f is regular guarantees that $f(T) \leq f(R) \leq f(S)$. ■

Chapter 4

Two-machine flowshop

In the rest of this dissertation we shall deal with the two-machine flowshop only. To simplify the notation we replace p_{1j} by a_j and p_{2j} by b_j for all $1 \leq j \leq n$, and moreover we assume without loss of generality that the jobs are numbered according to a nondecreasingly sorted processing times on the second machine, i.e that $b_1 \leq b_2 \leq \cdots \leq b_n$. Such an order is often called a Shortest Processing Time (or simply SPT) order. Now we are ready to prove a theorem which is a special case of a more general proposition from [3].

Theorem 1 *The set of all permutation schedules* $\mathbf{S}$ *is a dominant set for both* $F2||\sum C_j$ *and* $F2|nmit|\sum C_j$ *flowshop problems.*

Proof. Since $\mathcal{S}$ is a dominant set it is enough to prove that for every $S \in \mathcal{S}$ there exists a permutation schedule $S' \in \mathbf{S}$ such that $\sum_{j=1}^{n} C_j^{S'} \leq \sum_{j=1}^{n} C_j^{S}$. Note that the set $\mathcal{S}$ (and hence also the set $\mathbf{S}$) is different for each of the two problems under consideration. This is caused by the different method of constructing the $S_{\pi_1,\ldots,\pi_m}$ type schedules, depending on whether idle times are or are not allowed. However, the following simple interchange argument is the same for both cases and the difference between the sets $\mathcal{S}$ plays no role in it. So let $S \in \mathcal{S} \setminus \mathbf{S}$ be an arbitrary schedule. Since $S \notin \mathbf{S}$ there exists a pair of jobs J_i and J_j such that O_{1j} is scheduled immediately after O_{1i} in the schedule S (recall that for all schedules in $\mathcal{S}$ there is no idle time on the first machine regardless of whether idle time is allowed or not) while O_{2i} is scheduled after O_{2j} in the schedule S (possibly with an idle time and/or other jobs scheduled in between).

Now it is obvious that interchanging the position of O_{1i} and O_{1j} leads again to a feasible schedule. Moreover, such an interchange does not affect the value of the objective function because no operation on the second machine moves. Quite clearly, after a finite number of the above described interchanges we arrive to a permutation schedule S' which has the same value of the objective function as S. This finishes the proof. ∎

As we indicated in the introduction, the part of the paper [1] which deals with the $F2|nmit|\sum C_j$ problem contains incorrect claims. Let us show that this is indeed the case. Adiri and Pohoryles [1] claim that the optimal schedules have the following properties.

1. If in the optimal schedule for $F2|nmit|\sum C_j$, the blocking job is the last one, then the optimal schedule is according to SPT on the second machine, except for the blocking job that is the one with the minimum processing time on the second machine.

2. The jobs that (i) precede (ii) succeed the blocking job in the optimal schedule for $F2|nmit|\sum C_i$ are ordered according to SPT on M_2, provided the no-idle constraint is not violated.

The following example shows that neither of these claims holds, even if we restrict the problem by requiring that all processing times on the first machine are the same.

Example 2 Consider the 3-job instance with $a_1 = a_2 = a_3 = 4$, $b_1 = b_2 = 1, b_3 = 6$. Since $b_1 = b_2$, it suffices to consider three permutations only, for example, the permutations $\langle 1, 2, 3\rangle$, $\langle 1, 3, 2\rangle$ and $\langle 3, 1, 2\rangle$. The corresponding permutation schedules are given in Figure 4.1. Obviously the schedule corresponding to the permutation $\langle 3, 1, 2\rangle$ is optimal.

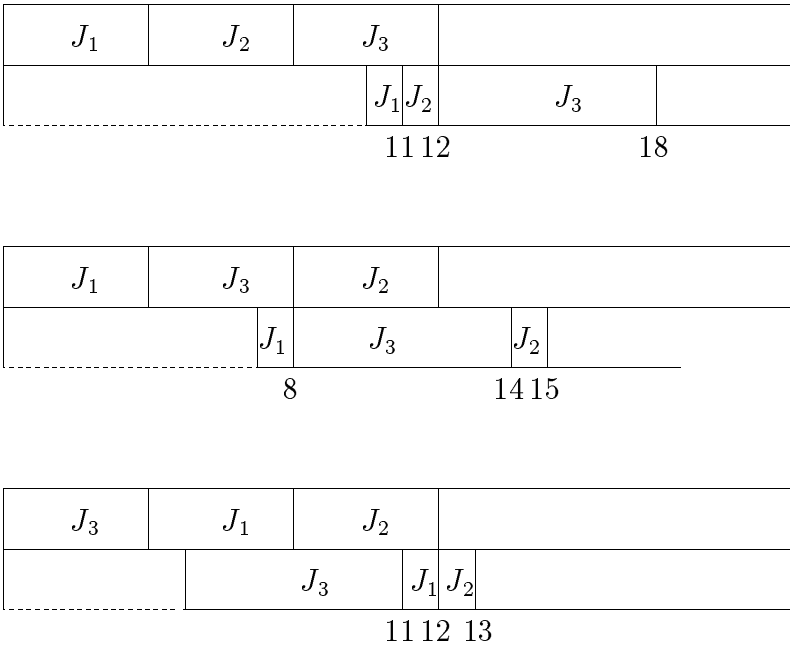

Figure 4.1:

Its blocking job is the last one, but its first two jobs are not in an SPT order, which contradicts the claims.

Now we shall prove that although the first claim is incorrect, the special schedule mentioned in the claim does have an interesting property. Let $\mathbf{S}^k$ denote the set of all schedules from $\mathbf{S}$ whose blocking position is k, $1 \leq k \leq n$, and let $\mathbf{I}^k$ denote the set of $n!$ schedules, both feasible and infeasible, defined as follows: for each permutation π, create the schedule in $\mathbf{I}^k$ by scheduling the jobs on both machines in the order prescribed by π (with no idle time), and then aligning the completion time of the operation O_{1k} with the starting time of the operation O_{2k}. Furthermore, let S^* be the schedule (possibly infeasible) from $\mathbf{I}^n$ corresponding to the permutation such that the last job is the one with the shortest operation on the second machine and the remaining jobs are ordered in an SPT order on the second machine.

Lemma 1 *S^* is best in $\mathbf{I}^n$.*

Proof. Let S be an arbitrary schedule from $\mathbf{I}^n$, π be the permutation defining the schedule S, and let t_1 denote the starting time of the first job in S on the second machine. It is

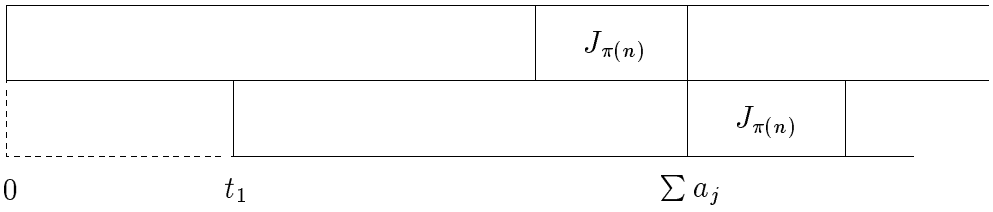

Figure 4.2:

easy to verify (see Figure 4.2) that

$$t_1 = \sum_{j=1}^{n} a_{\pi(j)} - \sum_{j=1}^{n-1} b_{\pi(j)} = \sum_{j=1}^{n} a_j - \sum_{j=1}^{n} b_j + b_{\pi(n)}.$$

Since the completion time $C^S_{\pi(j)}$ of the job in the jth position of schedule S can be calculated by

$$C^S_{\pi(j)} = t_1 + \sum_{i=1}^{j} b_{\pi(i)}.$$

We have

$$\sum_{j=1}^{n} C^S_j = \sum_{j=1}^{n} C^S_{\pi(j)} = nt_1 + \sum_{j=1}^{n} (n-j+1) b_{\pi(j)}.$$

Now the substitution for t_1 gives

$$\sum_{j=1}^{n} C^S_j = n(\sum_{j=1}^{n} a_j - \sum_{j=1}^{n} b_j) + (n+1)b_{\pi(n)} + nb_{\pi(1)} + (n-1)b_{\pi(2)} + \cdots + 2b_{\pi(n-1)}$$

where the first term of the sum on the right-hand side does not depend on the order of jobs. The remaining sum on the right hand side takes a minimum value if the permutation π corresponds to the schedule S^* which finishes the proof. ■

Proposition 4 *If S^* is feasible then it is optimal.*

Proof. Let $S \in \mathbf{S}$ be an arbitrary permutation schedule and let $S' \in \mathbf{I}^n$ be the schedule defined by the same permutation as S. Obviously, S and S' are identical on the first machine. Since S is feasible, the last operation on M_2 does not start before the conclusion of the last operation on M_1. Therefore S and S' are identical also M_2 or S' can be produced from S by shifting the schedule on M_2 to the left. Thus $\sum_{j=1}^{n} C^{S'}_j \leq \sum_{j=1}^{n} C^S_j$. However, by Lemma 1 $\sum_{j=1}^{n} C^{S^*}_j \leq \sum_{j=1}^{n} C^{S'}_j$ and therefore S^* is optimal. ■

Remark 1 If $a_i \geq b_j$ whenever $i \neq j$, then $S^* \in \mathbf{S}^n$, and we can conclude that S^* is optimal. In particular, if $M_1 \text{ -> } M_2$ then S^* is optimal. The latter is a special case Theorem 4 of [1], which deals with a decreasing series of dominating machines.

It should be pointed out that an argument similar to Lemma 1 cannot be applied to blocking position k with $k < n$, because then the sum $\sum_{j=1}^{k} a_{\pi(j)}$ depends on permutation π. Consequently the starting time of the blocking job on the second machine depends on the order of jobs on the first machine. This obstacle does not occur when the processing times on the first machine are all equal to a prescribed positive number a. Before considering this special case, we illustrate what type of results can hold for $k < n$ in general case.

Let us consider the case $k = 1$. Let S be an arbitrary schedule from $\mathbf{S}^1$ and let π be the permutation defining schedule S. Completion time $C^S_{\pi(i)}$ of the job in the ith position of S is obviously given by

$$C^S_{\pi(i)} = a_{\pi(1)} + \sum_{r=1}^{i} b_{\pi(r)}.$$

Thus the sum of all completion times resulting from schedule S can be computed as follows:

$$\sum_{i=1}^{n} C^S_{\pi(i)} = na_{\pi(1)} + \sum_{r=1}^{n} (n - r + 1) b_{\pi(r)}.$$

If the job in the first position of S is job J_j, then we obtain

$$\sum_{i=1}^{n} C^S_{\pi(i)} = na_j + nb_j + \sum_{r=2}^{n} (n - r + 1) b_{\pi(r)}.$$

Obviously, this sum is minimized when all the remaining jobs (i.e. all jobs except of J_j) are scheduled in an SPT order on the second machine. Let $(SPT)_j$ denote the schedule for which this minimum is attained and let $\mathbf{S}^1_j$ be the set of all schedules from $\mathbf{S}^1$ whose first job is J_j. By the above consideration it follows that if $(SPT)_j$ belongs to $\mathbf{S}^1_j$, then it is best in $\mathbf{S}^1_j$. Now let $S = (SPT)_j$, i.e. let π be the permutation defining $(SPT)_j$. The completion time $C^j_{\pi(i)}$ of the job in the ith position of $(SPT)_j$ can be computed as follows (see Figure 4.3)

$$C^j_{\pi(i)} = \begin{cases} a_j + b_j + \sum_{r=1}^{i-1} b_r & \text{for } i < j, \\ a_j + \sum_{r=1}^{i} b_r & \text{for } i \geq j. \end{cases}$$

Consequently

$$\sum_{i=1}^{n} C^j_i = \sum_{i=1}^{n} C^j_{\pi(i)} = na_j + (j-1)b_j - \sum_{i=1}^{j-1} b_i + \sum_{i=1}^{n} (n - i + 1) b_i.$$

Since the last term is independent of j, we can conclude that the best value is obtained for j at which the minimum value of

$$V_j := na_j + (j-1)b_j - \sum_{i=1}^{j-1} b_i$$

is attained. Let j^* be such that

$$V_{j^*} = \min_{i=1}^{n} V_j$$

where the minimum is taken over all j such that $\mathbf{S}^1_j \neq \emptyset$. Since

$$\mathbf{S}^1 = \mathbf{S}^1_1 \cup \mathbf{S}^1_2 \cup \cdots \cup \mathbf{S}^1_n = \bigcup_{\{j | \mathbf{S}^1_j \neq \emptyset\}} \mathbf{S}^1_j,$$

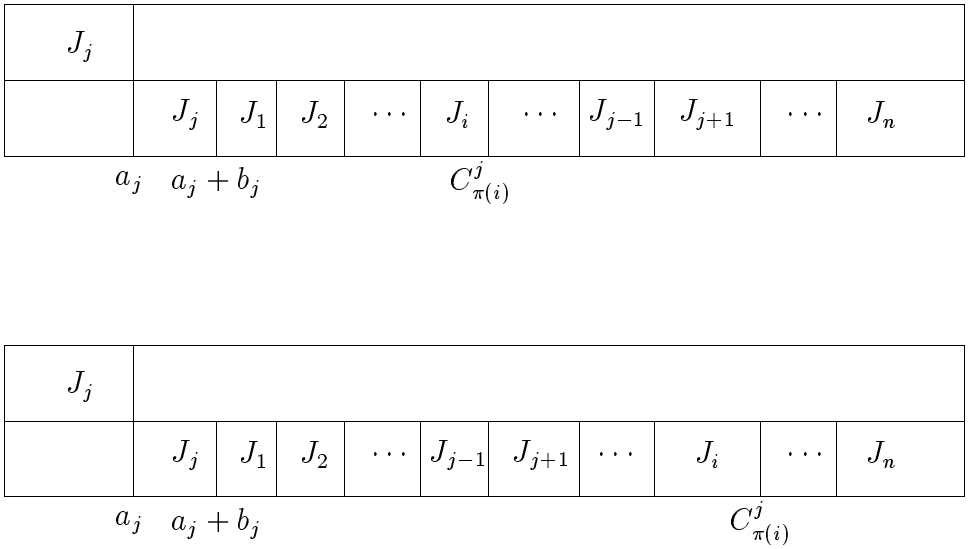

Figure 4.3:

we conclude that $V_{j^*} + \sum_{i=1}^{n}(n-i+1)b_i$ is the minimum value of the objective function over $\mathbf{S}^1$, provided $(SPT)_j$ belongs to $\mathbf{S}^1_j$ whenever $\mathbf{S}^1_j \neq \emptyset$.

Remark 2 If $a_i \leq b_j$ whenever $i \neq j$, then all sets $\mathbf{S}^1_j$ are nonempty and $(SPT)_j$ belongs to $\mathbf{S}^1_j$ for each j. Consequently $(SPT)_{j^*}$ is best in $\mathbf{S}^1$. Moreover, $\mathbf{S}^2 = \cdots = \mathbf{S}^n = \emptyset$ in this case. Therefore $(STP)_{j^*}$ is best in $\mathbf{S}$. In other words, $(SPT)_{j^*}$ is optimal. In particular, if $M_2 \rightarrow M_1$ then $(SPT)_{j^*}$ is optimal. This is a special case of Theorem 3 of the Adiri and Pohoryles paper [1] which deals with an increasing series of dominating machines.

From now on let us consider the special case in which the processing times on the first machine are all equal to a given positive integer a, or formally let $a_1 = a_2 = \cdots = a_n = a$. Moreover, we shall assume that $\min_{i=1}^{n} b_i < a < max_{i=1}^{n} b_i$, because otherwise the problem is easily solvable due to Remark 2 or Remark 1. For each k, $1 \leq k \leq n$, and each permutation π of jobs, let S^k_π denote the schedule (possibly infeasible) constructed as follows. On the first machine, the jobs are scheduled as in the corresponding permutation schedule. On the second machine, the jobs are scheduled in the same order but so that job $J_{\pi(k-1)}$ is completed at time $t = ka$ and no machine idle time exists between consecutive jobs (see Figure 4.4).

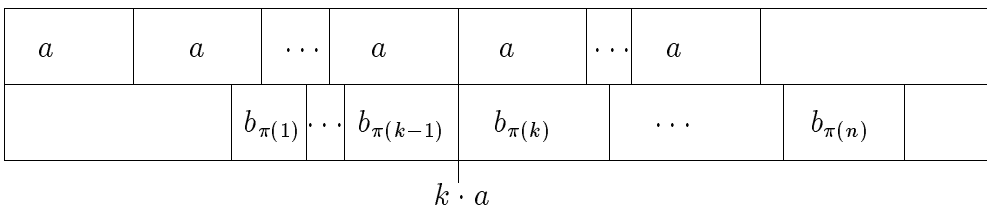

Figure 4.4:

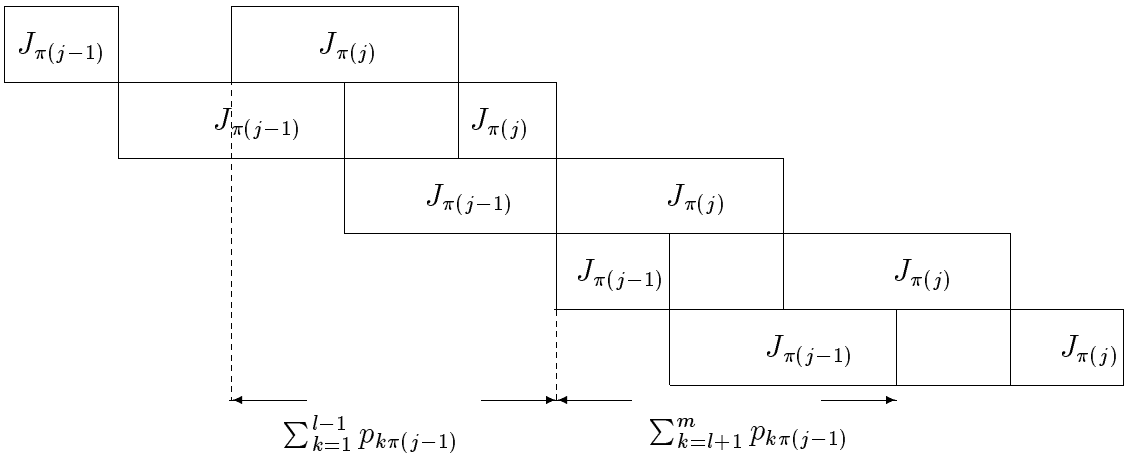

Figure 4.5:

Let $C^k_{\pi(i)}$ denote completion time of $J_{\pi(i)}$ in S^k_π. Then

$$C^k_{\pi(i)} = \begin{cases} ka - \sum_{j=i+1}^{k-1} b_{\pi(j)} & \text{for } 1 \le i \le k-1 \\ ka + \sum_{j=k}^{i} b_{\pi(j)} & \text{for } k \le i \le n \end{cases}$$

and the sum of completion times under schedule S^k_π can be computed as follows:

$$\sum_{i=1}^{n} C^k_{\pi(i)} = nka - \sum_{i=2}^{k-1}(i-1)b_{\pi(i)} + \sum_{i=k}^{n}(n-i+1)b_{\pi(i)}.$$

Since $k\sum_{j=1}^{n} b_j = k\sum_{i=1}^{n} b_{\pi(i)}$, we have

$$\begin{aligned} \sum_{i=1}^{n} C^k_{\pi(i)} &= k(na - \sum_{j=1}^{n} b_j) - \sum_{i=2}^{k-1}(i-1)b_{\pi(i)} + \sum_{i=k}^{n}(n-i+1)b_{\pi(i)} + k\sum_{i=1}^{n} b_{\pi(i)} \\ &= k(na - \sum_{j=1}^{n} b_j) + \sum_{i=1}^{k-1}(k-i+1)b_{\pi(i)} + \sum_{i=k}^{n}(n+k-i+1)b_{\pi(i)}. \qquad (4.1) \end{aligned}$$

The term $k(na - \sum_{j=1}^{n} b_j)$ does not depend on the order of jobs. Note that in the remaining two summations each $b_{\pi(i)}$ appears different number of times: $b_{\pi(k-1)}$ is in the sum just twice (and therefore the longest job J_n should be scheduled in the $(k-1)$th position), $b_{\pi(k-2)}$ three times, and so on until $b_{\pi(1)}$ appears k times. In the second summation $b_{\pi(n)}$ is present $k+1$ times, $b_{\pi(n-1)}$ $k+2$ times, and so on until finally $b_{\pi(k)}$ appears $n+1$ times. Therefore the minimum value of (4.1) is achieved for the permutation constructed as follows. First, order the jobs in a nondecreasing order of b_i's (SPT order). Then schedule the first segment of $n-k+1$ jobs from $t = ka$ and place the remaining segment of $k-1$ jobs in front of $t = ka$ without allowing any machine idle time between consecutive jobs. Since we have assumed that the jobs are numbered in the nondecreasing order of b_i's, the resulting job order is

$$\hat{\pi} = \langle J_{n-k+2}, J_{n-k+3}, \cdots, J_n, J_1, J_2, \cdots, J_{n-k+1} \rangle.$$

Of course the corresponding schedule $S^k_{\hat{\pi}}$ is not necessarily in $\mathbf{S}^k$. Because all schedules from $\mathbf{S}^k$ have the form S^k_π for some π, the schedule $S^k_{\hat{\pi}}$ provides a lower bound for $\mathbf{S}^k$. As a consequence, we have the following proposition.

Proposition 5 If $S^k_{\hat{\pi}}$ belongs to $\mathbf{S}^k$, then it is best in $\mathbf{S}^k$.

It should be pointed out that this proposition is of very limited value, because as a rule $S^k_{\hat{\pi}}$ doesn't belong to $\mathbf{S}^k$. In particular, if $1 < k < n$, then $b_n \leq a$ must hold for the job J_n to start only after its operation on the first machine is completed, and similarly $a \leq b_1$ must hold so that the job J_2 starts only after its operation on the first machine is completed. Therefore $S^k_{\hat{\pi}}$ belongs to $\mathbf{S}^k$ if and only if $max_{1\leq i\leq n} b_i \leq a \leq min_{1\leq i\leq n} b_i$. However, it happens if and only if $b_1 = b_2 = \cdots = b_n = a$, and then every permutation schedule is optimal. The usefulness of schedule $S^k_{\hat{\pi}}$ is in the fact that it provides a lower bound for $\mathbf{S}^k$.

A further extension of this simple bound and Proposition 5 will be given in the following chapter in connection with the branch-and-cut technique.

Chapter 5

Branch-and-Cut technique

In this chapter, we describe the proposed branch-and-cut procedures for solving the problem $F2|p_{1j} = a, nmit| \sum C_j$. We assume, without loss of generality, that

$$b_1 \leq b_2 \leq \cdots \leq b_n \text{ and } b_1 < a < b_n.$$

In general terms, we can describe the procedure as follows. At any stage of our search the set of all permutation schedules is partitioned into a family of subsets. Each of these subsets represents an active subproblem consisting of minimizing the objective function on that subset. The strategy is to select some active subproblem, examine it, and decide whether or not any solution to that subproblem needs to be considered further in identifying an optimal solution of the original problem. If not, we consider that subproblem fathomed, and we remove it from the list of active subproblems. If it cannot be fathomed, then it is replaced in the list of active subproblems by one or more new active subproblems each of which is derived by imposing some additional constraints. The entire process can be represented as a tree where nodes in the tree correspond to subproblems and where branches are formed according to restrictions that are imposed for creating the subproblems

In the following subsections we specify the general strategy in more detail. For ease of presentation, we refine the previous notation as follows. A permutation of a subset of jobs is called a partial sequence. If A and B are partial sequences of two disjoint subsets of jobs, then $S^k(B, A)$ denotes the set of all schedules from $\mathbf{S}^k$ in which B is completed at time ka and A is scheduled immediately after B - see Figure 5.1. In this notation, set $\mathbf{S}^k$ becomes $S^k(\Lambda, \Lambda)$ where Λ stands for the partial sequence of empty set. The problem of

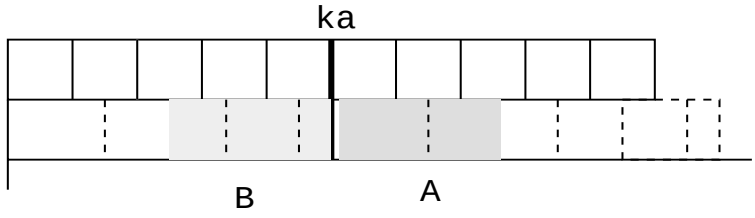

Figure 5.1:

minimizing the sum of completion times over $S^k(B, A)$ will be called subproblem $P^k(B, A)$.

If $S^k(B, A)$ is empty, then we say that $P^k(B, A)$ is infeasible, otherwise it is feasible. If A is a partial sequence, then $|A|$ denotes the number of elements of the corresponding set.

5.1 Feasibility

During the run of the branch-and-cut algorithm subproblems need to be tested for feasibility. This can be done as follows. Let A and B be partial sequences of two disjoint subsets of jobs and let k be a given integer, $1 \leq k \leq n$. First obvious condition for a feasibility of subproblem $P^k(B, A)$ is

$$0 \leq |B| \leq k-1 \text{ and } 0 \leq |A| \leq n-k+1. \tag{5.1}$$

To formulate other conditions, suppose that $A = \langle J^A_1, J^A_2, \cdots, J^A_{|A|} \rangle$, and let b^A_i denote the processing time of J^A_i on the second machine. Since the jobs of A must be scheduled from time ka without any inserted machine idle time, the following system of inequalities must be satisfied.

$$\sum_{i=1}^{r} b^A_i \geq ra \quad \text{for} \quad 1 \leq r \leq |A| - 1 \tag{5.2}$$

If $|A| < n-k+1$, then the previous inequality must hold also for $r = |A|$, i.e

$$\sum_{i=1}^{|A|} b^A_i \geq |A|a. \tag{5.3}$$

Analogously, let $B = \langle J^B_{|B|}, J^B_{|B|-1}, \cdots, J^B_1 \rangle$ and let b^B_i denote the processing time of J^B_i on the second machine. Then, for the feasibility of $P^k(B, A)$, the following system of inequalities must hold:

$$\sum_{i=1}^{r} b^B_i < ra \quad \text{for} \quad 1 \leq r \leq |B|. \tag{5.4}$$

The strict inequalities are required, because if the equality holds for some r, then the resulting schedules does not belong to $S^k(B, A)$ but to $S^{k-r}(C, D)$ where

$$C = \langle J^B_{|B|}, J^B_{|B|-1}, \cdots, J^B_{r+1} \rangle,$$

$$D = \langle J^B_r, J^B_{r-1}, \cdots, J^B_1, A \rangle.$$

For easy reference, we say that partial sequence A is right-feasible for position k, when (5.2) and, if applicable, also (5.3) are satisfied. Similarly, we say that B is left-feasible for position k, when (5.4) is satisfied. If A is right-feasible and B is left-feasible for position k, then we say that $\langle B, A \rangle$ is feasible for position k.

If $\langle B, A \rangle$ is feasible for position k, then there is a chance that partial sequence $\langle B, A \rangle$ may be extended into a complete sequence which determines a schedule belonging to $S^k(B, A)$. To see what conditions must be further fulfilled, we first observe that $k-1-|B|$ jobs must be scheduled in front of B and $n-k+1-|A|$ jobs must be scheduled after A. Let n_A, n_B, M_A and M_B be defined as follows:

$$n_A := n - k + 1 - |A|, \quad n_B := k - 1 - |B|,$$
$$M_A := \sum_{i=1}^{|A|} b_i^A - |A|a, \quad M_B := |B|a - \sum_{i=1}^{|B|} b_i^B.$$

Further, let A' and B' be partial sequences of n_A and n_B jobs belonging neither to A nor to B. For definiteness suppose that

$$A' = \langle J_1^{A'}, J_2^{A'}, \cdots, J_{n_A}^{A'} \rangle, \quad B' = \langle J_{n_B}^{B'}, J_{n_B-1}^{B'}, \cdots, J_1^{B'} \rangle.$$

It is easy to verify that if the permutation schedule defined by $\langle B', B, A, A' \rangle$ belongs to $S^k(B, A)$, then

$$\sum_{i=1}^{r} b_i^{B'} < ra + M_B \quad \text{for} \quad 1 \le r \le n_B, \tag{5.5}$$

$$\sum_{i=1}^{r} b_i^{A'} \ge ra - M_A \quad \text{for} \quad 1 \le r \le n_A - 1. \tag{5.6}$$

Obviously, if these conditions are not satisfied for A' and B' such that

$$b_1^{A'} \ge b_2^{A'} \ge \cdots \ge b_{n_A}^{A'} \ge b_{n_B}^{B'} \ge b_{n_B-1}^{B'} \ge \cdots \ge b_1^{B'},$$

then they cannot be satisfied for any other choice of A' and B'. Consequently, in such cases $S^k(B, A)$ is empty and problem $P^k(B, A)$ is infeasible.

Example 3 Consider the instance of 7 jobs with $a = 30$ and b_j given by the following Table 5.1.

Table 5.1:

j	1	2	3	4	5	6	7
b_j	29	31	32	33	33	36	40

1. Consider $P^3(\Lambda, \langle J_1, J_2 \rangle)$. In this case condition (5.2) is not satisfied for $r = 1$, because
$$\sum_{i=1}^{r} b_i^A = b_1^A = b_1 = 29 < 30 = ra.$$
Therefore $P^3(\Lambda, \langle J_1, J_2 \rangle)$ is infeasible.

2. Consider $P^4(\langle J_1 \rangle, \langle J_4 \rangle)$. Now all conditions (5.2)-(5.4) are satisfied. It remains to verify conditions (5.5) and (5.6) for $A' = \langle J_1^{A'}, J_2^{A'}, J_3^{A'} \rangle$ and $B' = \langle J_2^{B'}, J_1^{B'} \rangle$ such that
$$b_1^{A'} \ge b_2^{A'} \ge b_3^{A'} \ge b_2^{B'} \ge b_1^{B'}$$
where all jobs are different from J_1 and J_4. It follows that $A' = \langle J_7, J_6, J_5 \rangle$ and $B' = \langle J_3, J_2 \rangle$ Now it is easy to verify that conditions (5.6) are satisfied, because
$$\begin{aligned} b_7 &= 40 \ge 30 - 3 = a - M_A, \\ b_6 + b_7 &= 76 \ge 2 \cdot 30 - 3 = 2a - M_A. \end{aligned} \tag{5.7}$$

However, conditions (5.5) are not fulfilled because

$$b_2 \;=\; 31 = 30 + 1 = a + M_B.$$

Strict inequality is required instead of the above equality and therefore we again conclude that $P^4(\langle J_1\rangle, \langle J_4\rangle)$ is infeasible.

3. Consider $P^k(\Lambda, \Lambda)$ for $1 \le k \le 7$. First we observe that $P^k(\Lambda, \Lambda)$ is infeasible for $4 \le k \le 7$. This follows from the fact that (5.5) cannot be satisfied, because $M_B = 0$ and $b_1 + b_2 + b_3 > 3a$. Next we observe that $P^3(\Lambda, \Lambda)$ is also infeasible because we must have $b_1^{B'} < a$ and $b_2^{B'} + b_1^{B'} < 2a$, the latter being impossible. Finally we observe that the schedule given by $\langle J_n, J_{n-1}, \cdots J_1\rangle$ belongs to $S^1(\Lambda, \Lambda)$ and the schedule given by $\langle J_1, J_n, J_{n-1}, \cdots J_2\rangle$ belongs to $S^2(\Lambda, \Lambda)$. Therefore both $P^1(\Lambda, \Lambda)$ and $P^2(\Lambda, \Lambda)$ are feasible.

5.2 Lower bounds

During initialization and fathoming some easily computable lower bounds on the value of optimal solutions of subproblems should be available. In section 4 we have derived simple lower bounds on the value of the objective function over $S^k(\Lambda, \Lambda)$. In the simplest version of our procedure we use the following extension of these simple bounds to arbitrary $S^k(B, A)$.

5.2.1 Simple lower bounds

Let $\langle B, A\rangle$ be feasible for blocking position k. Consider the corresponding subproblem $P^k(B, A)$ and its feasible set $S^k(B, A)$. Let π be a permutation constructed from the partial sequence $\langle B, A\rangle$ by completing it to a complete sequence as follows. From among the jobs that do not appear in $\langle B, A\rangle$ put n_B jobs before B and n_A jobs after A, in both cases in an arbitrary order. Having π, we associate with it the following, possibly infeasible, schedule S. On both machines the jobs are scheduled as in the corresponding permutation schedule but on the second machine the schedule is shifted so that the job in the $\pi(k-1)$th position is completed at time $t = ka$. Let $C^k_{\pi(i)}$ denote completion time of $J_{\pi(i)}$ under S. Using (4.1) we obtain

$$\begin{aligned}
\sum_{i=1}^{n} C^k_{\pi(i)} &= k(na - \sum_{j=1}^{n} b_j) + \sum_{i=1}^{k-1}(k-i+1)b_{\pi(i)} + \sum_{i=k}^{n}(n+k-i+1)b_{\pi(i)} \\
&= k(na - \sum_{j=1}^{n} b_j) + \sum_{i=1}^{k-1-|B|}(k-i+1)b_{\pi(i)} + \sum_{i=k-|B|}^{k-1}(k-i+1)b_{\pi(i)} \\
&\quad + \sum_{i=k}^{k+|A|-1}(n+k-i+1)b_{\pi(i)} + \sum_{i=k+|A|}^{n}(n+k-i+1)b_{\pi(i)}. \qquad (5.8)
\end{aligned}$$

The term $k(na - \sum_{j=1}^{n} b_j)$ does not depend on the order of jobs, and the terms $\sum_{i=k-|B|}^{k-1}(k-i+1)b_{\pi(i)}$ and $\sum_{i=k}^{k+|A|-1}(n+k-i+1)b_{\pi(i)}$ do not depend on the order of jobs outside of

$\langle B, A\rangle$. It follows from the remaining terms that the minimum value of (5.8) is achieved for the permutation constructed as follows.

First order all jobs not appearing in $\langle B, A\rangle$ according to SPT. Let A' denote the initial segment of n_A jobs of this order and let B' denote the remaining segment of n_B jobs. Then order the jobs according to the sequence $\sigma(B, A) = \langle B', B, A, A'\rangle$ and schedule them so that the completion time of the last job in B on the second machine is kept at ka (i.e. so that k is the blocking position) and there is no machine idle time between the consecutive jobs (See Figure 5.2). Since all schedules from $S^k(B, A)$ are constructed as described at

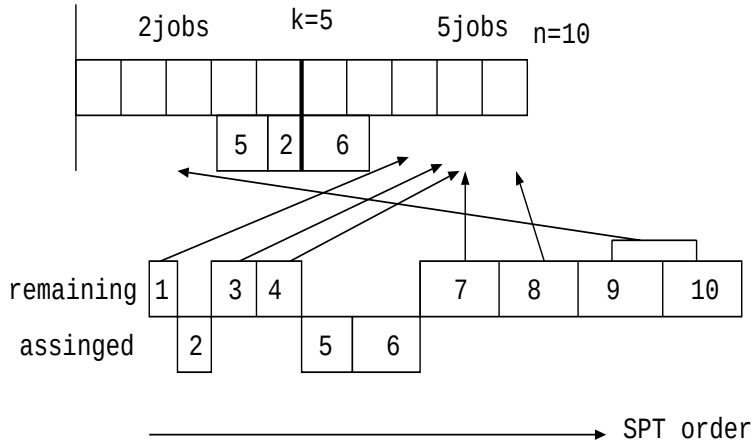

Figure 5.2: A permutation for getting lower bound of $P^5(\langle J_5, J_2\rangle, \langle J_6\rangle)$.

the beginning of this subsection, the schedule corresponding to $\hat{\pi}$ gives a lower bound for $S^k(B, A)$, and if it belongs to $S^k(B, A)$, it is best in $S^k(B, A)$.

5.2.2 Improved lower bounds

The simple lower bounds of the previous subsection can be improved in several ways. In our computational experiments, we have used an improvement based on the following idea, which we call "pick up and shift". Consider a feasible subproblem $P^k(B, A)$ where $\langle B, A\rangle$ is feasible for position k. The simple lower bound described in the previous subsection is computed as the value of schedule S defined by $\sigma(B, A) = \langle B', B, A, A'\rangle$ with the blocking position k. If S is feasible, then it is the best one in $P^k(B, A)$, and no improvement is possible. If S is infeasible, then there is a chance for improvement of the lower bound for $P^k(B, A)$. The infeasibility of S can be caused either by the fact that the partial sequence $\langle A, A'\rangle$ is not right-feasible for position k or $\langle B', B\rangle$ is not left-feasible for position k (or both). Without loss of generality let us assume that $\langle A, A'\rangle$ is not right-feasible for position k (the other case is symmetric) and let us denote $A' = \langle J_1^{A'}, \ldots, J_{n_A}^{A'}\rangle$. Let q, $1 \leq q < n_A$, be the smallest integer such that $\langle A, J_q^{A'}\rangle$ is right-feasible for k. If no such q exists, then set $q = n_A$.

Lemma 2 *Let q be defined as above and let S'_q be the schedule (possibly infeasible) defined by the sequence $\sigma(B, \langle A, J_q^{A'}\rangle)$ and such that k is its blocking position. Then the objective function value of S'_q provides a lower bound for $P^k(B, A)$.*

Proof Let J_p be an arbitrary job not in $A \cup B$ and let S_p be the schedule (possibly infeasible) defined by the sequence $\sigma(B, \langle A, J_p \rangle)$ and such that k is its blocking position. First let us note that S_p provides a lower bound for $P^k(B, \langle A, J_p \rangle)$. This follows from the fact that by definition of $\sigma(B, \langle A, J_p \rangle)$ the jobs not in $A \cup B \cup \{J_p\}$ are ordered in S_p in an SPT order, just like if constructing the simple lower bound for $P^k(B, \langle A, J_p \rangle)$. Since the subproblems $P^k(B, \langle A, J_p \rangle)$ for all choices of $J_p \notin A \cup B$ constitute a partition of $P^k(B, A)$, it will be enough to prove that S'_q is not worse than S_p for every $J_p \neq J_q^{A'}$. Due to the choice of q we get that for every p such that $b_p < b'_q$ (where b'_q is the processing time of the job $J_q^{A'}$ on the second machine) the subproblem $P^k(B, \langle A, J_p \rangle)$ is infeasible. Therefore it will be enough to show that the objective function for S_q is not larger than for S_p whenever $b_p \geq b'_q$. So let J_p be an arbitrary job not in $A \cup B$ such that $b_p \geq b'_q$. We shall distinguish two cases.

- Let $J_p \in A'$. Therefore we can denote $J_p = J_r^{A'}$ for some $r > q$ (in this notation $S_p = S'_r$ and $b_p = b'_r$). Moreover let R_i denotes the conclusion time of the job $J_i^{A'}$ in the schedule S'_r and Q_i the conclusion time of the job $J_i^{A'}$ in the schedule S'_q. Then

$$\begin{aligned}
\sum_{i=1}^{n} C_i^{S'_r} - \sum_{i=1}^{n} C_i^{S'_q} &= (\sum_{i=1}^{q-1} R_i - \sum_{i=1}^{q-1} Q_i) + (R_q - Q_q) \\
&+ (\sum_{i=q+1}^{r-1} R_i - \sum_{i=q+1}^{r-1} Q_i) + (R_r - Q_r) \\
&= (q-1)(b'_r - b'_q) + (b'_r + \sum_{i=1}^{q-1} b'_i) \\
&+ (r-q-1)b'_r - (b'_q + \sum_{i=1}^{q-1} b'_i + \sum_{i=q+1}^{r-1} b'_i) \\
&= q(b'_r - b'_q) + \sum_{i=q+1}^{r-1} (b'_r - b'_i) \geq 0. \qquad (5.9)
\end{aligned}$$

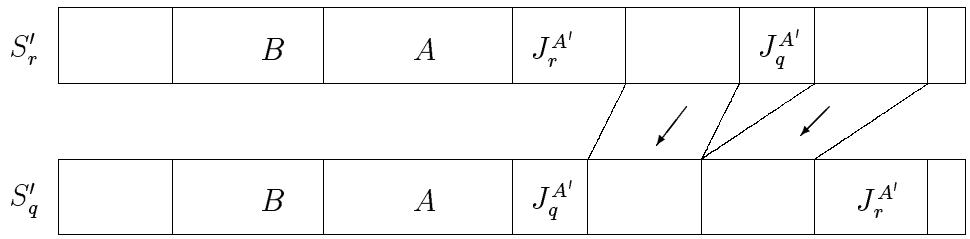

Figure 5.3:

- Let $J_p \in B'$. Denote $B' = \langle J_{n_A+1}^{A'}, \ldots, J_{n_A+n_B}^{A'} \rangle$ and let $J_p = J_r^{A'}$ for some $r > n_A$. Then

$$\sum_{i=1}^{n} C_i^{S'_r} - \sum_{i=1}^{n} C_i^{S'_q} = (\sum_{i=1}^{q-1} R_i - \sum_{i=1}^{q-1} Q_i) + (R_q - Q_q) + (\sum_{i=q+1}^{n_A-1} R_i - \sum_{i=q+1}^{n_A-1} Q_i)$$

$$
\begin{aligned}
&+ \quad (R_{n_A} - Q_{n_A}) + (\sum_{i=n_A+1}^{r-1} R_i - \sum_{i=n_A+1}^{r-1} Q_i) + (R_r - Q_r) \\
&= \quad (q-1)(b'_r - b'_q) + (b'_r + \sum_{i=1}^{q-1} b'_i) + (n_A - q - 1)b'_r - (\sum_{i=1}^{n} b_i - b'_{n_A}) \\
&+ \quad (r - n_A - 1)b'_r - (b'_q + b'_{n_A} + \sum_{i=1}^{q-1} b'_i + \sum_{i=q+1}^{n_A-1} b'_i + \sum_{i=n_A+1}^{r-1} b'_i - \sum_{i=1}^{n} b_i) \\
&= \quad q(b'_r - b'_q) + \sum_{i=q+1}^{n_A-1} (b'_r - b'_i) + \sum_{i=n_A+1}^{r-1} (b'_r - b'_i) \geq 0. \qquad (5.10)
\end{aligned}
$$

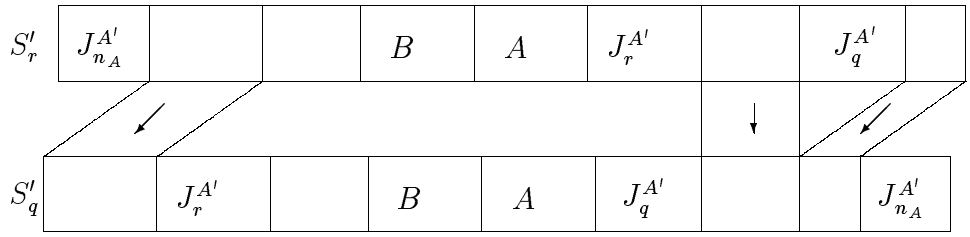

Figure 5.4:

■

Applying the same idea as above to the partial sequence $\langle B', B\rangle$ (if it is not left-feasible for position k), we obtain an analogous improvement corresponding to a sequence $\sigma(\langle J^{B'}_q, B\rangle, A)$ for some job $J^{B'}_q$. By picking the better of the two lower bounds we obtain the improved lower bound for the problem $P^k(B, A)$.

5.2.3 Lower bounds based on LP relaxation

It turns out that the problem $F2|p_{1j} = a, nmit|\sum C_j$ can be formulated as the following integer programming problem. Let y_0 denote the starting time of the first job on the second machine, y_k denote the completion time of the job in the kth position, $1 \leq k \leq n$, and finaly let x_{jk} be defined by

$$
x_{jk} = \begin{cases} 1 & \text{if } J_j \text{ is in the } k\text{th position} \\ 0 & \text{otherwise} \end{cases}
$$

Then the problem can be formulated as follows:

$$
\begin{array}{lll}
\text{minimize} & \sum_{k=1}^{n} y_k & \\
\text{subject to} & \sum_{j=1}^{n} x_{jk} = 1 & 1 \leq k \leq n \\
& \sum_{k=1}^{n} x_{jk} = 1 & 1 \leq j \leq n \\
& x_{jk} \in \{0, 1\} & 1 \leq j, k \leq n \\
& & \\
& y_k - y_{k-1} - \sum_{j=1}^{n} b_j x_{jk} = 0, & 1 \leq k \leq n \\
& y_{k-1} \geq ka. & 1 \leq k \leq n
\end{array}
$$

By relaxing the constraints $x_{jk} \in \{0,1\}$ to $0 \leq x_{jk} \leq 1$, we obtain a linear programming problem the optimal value of which gives a lower bound on the optimal value of the original problem. Since each subproblem $P^k(B,A)$ can be obtained by specifying values of some variables x_{jk} and y_k, we can use similar relaxations for calculating lower bounds for $P^k(B,A)$ and testing feasibility. Moreover, the above LP formulation has a hereditary property. That means that the formulation for subproblems has exactly the same structure as for the original problem, only on a smaller set of variables. By gradually fixing the values of variables, the inequalities with no variables left either turn into tautologies, or (in case of the inequalities of the type $y_{k-1} \geq ka$) turn into feasibility constraints for the given subproblem.

Example 4 Consider the subproblem $P^2(\Lambda, \langle J_3 \rangle)$ of a 4-job instance given by a, b_1, b_2, b_3, b_4. Then the relaxation of the original problem is the LP problem

$$
\begin{aligned}
\text{minimize} \quad & y_1 + y_2 + y_3 + y_4 \\
\\
\text{subject to} \quad & x_{11} + x_{21} + x_{31} + x_{41} = 1, \\
& x_{12} + x_{22} + x_{32} + x_{42} = 1, \\
& x_{13} + x_{23} + x_{33} + x_{43} = 1, \\
& x_{14} + x_{24} + x_{34} + x_{44} = 1, \\
& x_{11} + x_{12} + x_{13} + x_{14} = 1, \\
& x_{21} + x_{22} + x_{23} + x_{24} = 1, \\
& x_{31} + x_{32} + x_{33} + x_{34} = 1, \\
& x_{41} + x_{42} + x_{43} + x_{44} = 1, \\
& 0 \leq x_{jk} \leq 1, \qquad\qquad 1 \leq j,k \leq 4 \\
\\
& y_1 - y_0 - b_1 x_{11} - b_2 x_{21} - b_3 x_{31} - b_4 x_{41} = 0, \\
& y_2 - y_1 - b_1 x_{12} - b_2 x_{22} - b_3 x_{32} - b_4 x_{42} = 0, \\
& y_3 - y_2 - b_1 x_{13} - b_2 x_{23} - b_3 x_{33} - b_4 x_{43} = 0, \\
& y_4 - y_3 - b_1 x_{14} - b_2 x_{24} - b_3 x_{34} - b_4 x_{44} = 0, \\
& y_0 \geq a, \\
& y_1 \geq 2a, \\
& y_2 \geq 3a, \\
& y_3 \geq 4a.
\end{aligned}
$$

The LP relaxation of $P^2(\Lambda, \langle J_3 \rangle)$ is obtained by fixing job J_3 at the second position. This means that $x_{32} = 1$. It follows that $x_{3s} = x_{r2} = 0$ for all $r \neq 3$ and $s \neq 2$, and that

$y_1 = 2a, y_2 = 2a + b_3$. Therefore the resulting LP problem can be written as

$$
\begin{array}{lll}
\text{minimize} & y_3 + y_4 & \\
\\
\text{subject to} & x_{11} + x_{21} + x_{41} = 1, & \\
& x_{13} + x_{23} + x_{43} = 1, & \\
& x_{14} + x_{24} + x_{44} = 1, & \\
& x_{11} + x_{13} + x_{14} = 1, & \\
& x_{21} + x_{23} + x_{24} = 1, & \\
& x_{41} + x_{43} + x_{44} = 1, & \\
& 0 \le x_{jk} \le 1, & 1 \le j, k \le 4 \\
& & j \ne 3, k \ne 2 \\
& 2a - y_0 - b_1 x_{11} - b_2 x_{21} - b_4 x_{41} = 0, & \\
& y_3 - (2a + b_3) - b_1 x_{13} - b_2 x_{23} - b_4 x_{43} = 0, & \\
& y_4 - y_3 - b_1 x_{14} - b_2 x_{24} - b_4 x_{44} = 0, & \\
& y_0 \ge a, & \\
& y_3 \ge 4a. &
\end{array}
$$

provided that $2a + b_3 \ge 3a$. If the last inequality is not satisfied, i.e. if $b_3 < a$, then we conclude that $P^2(\Lambda, \langle J_3 \rangle)$ is infeasible, because the inequality $y_2 \ge 3a$ from the original LP formulation is not fulfilled.

5.3 Branching rule

When a subproblem cannot be fathomed, then further branching from the corresponding node is performed. Suppose that we must branch from the node representing $P^k(B, A)$. We may assume that $n_A + n_B > 0$, because otherwise a complete sequence can be obtained and no branching is necessary.

If $n_A \ne 0$ and $n_B \le n_A$, then we generate subproblems $P^k(B, \langle A, J_j \rangle)$ such that

- J_j belongs neither to A nor to B,
- $\langle A, J_j \rangle$ is right-feasible for position k.

If $n_B > n_A$, then we generate subproblems $P^k(\langle J_i, B \rangle, A)$ such that

- J_i belongs neither to A nor to B,
- $\langle J_i, B \rangle$ is left-feasible for position k.

Generally not all of the generated subproblems are placed in the list of active subproblems. Some of them may be discarded because infeasibility, some because their lower bounds are greater than or equal to the incumbent upper bound. It may also happen that a generated subproblem is completely solved. Then its solution is compared with the incumbent one and the better one is kept as the incumbent solution. Finally, some of the feasible subproblems may be discarded because they are dominated as explained in the following section.

5.4 Dominance rules

Suppose that we are branching from the node representing subproblem $P^k(B, A)$ and that we are to create subproblems $P^k(B, \langle A, J_j \rangle)$ according to the rules described in the previous section. As pointed out in the previous section, we can discard infeasible problems. Among feasible ones we may discard those whose lower bounds are greater or equal to the incumbent upper bound on the optimal value. In addition we may discard the complete ones after comparison with the incumbent solution. However, there may still be possible to discard some feasible subproblems, provided they are dominated in the following sense.

We say that a subproblem dominates another subproblem if the best complete schedule emanating from the former is no worse than the best schedule emanating from the latter.

Let us now describe the dominance rules we have used in our computational experiments.

5.4.1 Zero-gap dominance rules

Consider a feasible subproblem $P^k(B, \langle A, J_i, J_j \rangle)$. It can easily be seen that if $\langle A, J_j, J_i \rangle$ is right-feasible for position k, then $P^k(B, \langle A, J_j, J_i \rangle)$ dominates $P^k(B, \langle A, J_i, J_j \rangle)$ whenever $b_i \geq b_j$. Indeed, let S be an arbitrary schedule from $P^k(B, \langle A, J_i, J_j \rangle)$ and let S' be the schedule obtained from S by interchanging the order of J_i and J_j. It follows from the right-feasibility of $\langle A, J_j, J_i \rangle$ for k, that S' belongs to $P^k(B, \langle A, J_j, J_i \rangle)$.

Let t denote the starting time of J_i on the second machine under schedule S. Obviously (see Figure 5.5), we obtain

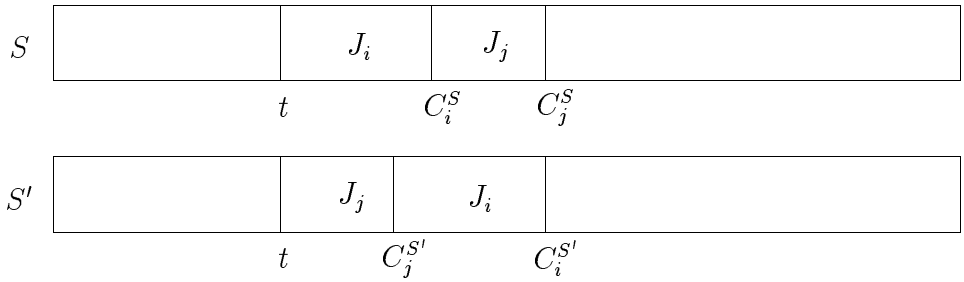

Figure 5.5:

$$\begin{aligned} \sum_{k=1}^{n} C_k^S - \sum_{k=1}^{n} C_k^{S'} &= C_i^S + C_j^S - (C_j^{S'} + C_i^{S'}) \\ &= C_i^S - C_j^{S'} \\ &= t + b_i - (t + b_j) \\ &= b_i - b_j. \end{aligned}$$

Since $b_i \geq b_j$, the sum of completion times under S' is better than under S. Consequently, $P^k(B, \langle A, J_i, J_j \rangle)$ is dominated by $P^k(B, \langle A, J_j, J_i \rangle)$ and therefore $P^k(B, \langle A, J_i, J_j \rangle)$ can be discarded. By analogous argument, we can show that a feasible subproblem

$P^k(\langle J_j, J_i, B\rangle, A)$ can be discarded, whenever $b_j \geq b_i$, provided $\langle J_i, J_j, B\rangle$ is left-feasible for k.

Similar dominance rules can be derived by interchanging pair of jobs or triples of jobs or longer tuples of jobs instead of interchanging single jobs. In order to distinguish these rules from those described in the next section we shall call them zero-gap rules and refer to them as D_{10}-rules, D_{20}-rules, and so on, where the first index indicates the length of the interchange tuples. For example, if $P^k(B, \langle A, J_r, J_s, J_i, J_j\rangle)$ and $P^k(B, \langle A, J_i, J_j, J_r, J_s\rangle)$ are feasible, then the former is dominated by the latter whenever $b_r + b_s \geq b_i + b_j$.

5.4.2 Nonzero-gap dominance rules

Consider feasible subproblem $P^k(B, \langle A, J_i, J_g, J_j\rangle)$. Analogously to the previous subsection, we show that this subproblem is dominated by $P^k(B, \langle A, J_j, J_g, J_i\rangle)$ whenever $b_i \geq b_j$, provided $\langle A, J_j, J_g, J_i\rangle$ is right-feasible for k. Let S belongs to $P^k(B, \langle A, J_i, J_g, J_j\rangle)$ and let S' be the schedule obtained by interchanging the jobs J_i and J_j and keeping J_g between them. Then we obtain a schedule from $P^k(B, \langle A, J_j, J_g, J_i\rangle)$ such that

$$\begin{aligned}
\sum_{k=1}^{n} C_k^S - \sum_{k=1}^{n} C_k^{S'} &= C_i^S + C_g^S + C_j^S - (C_j^{S'} + C_g^{S'} + C_i^{S'}) \\
&= C_i^S + C_g^S - (C_j^{S'} + C_g^{S'}) \\
&= t + b_i + t + b_i + b_g - (t + b_j + t + b_j + b_g) \\
&= 2(b_i - b_j)
\end{aligned}$$

where t denotes the starting time of J_i on the second machine in S (see Figure 5.6).

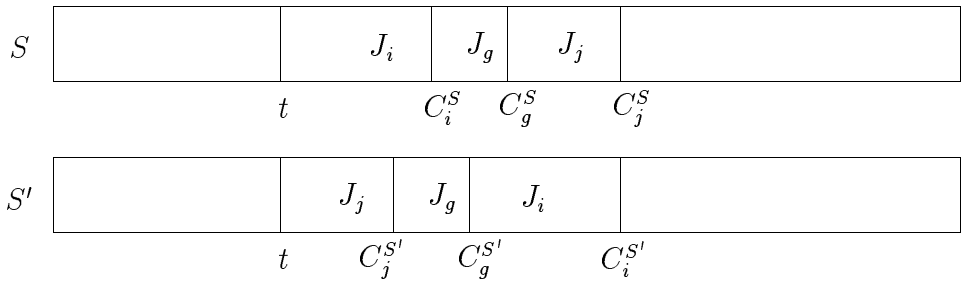

Figure 5.6:

Analogously we can show that if $P^k(\langle J_j, J_g, J_i, B\rangle, A)$ and $P^k(\langle J_i, J_g, J_j, B\rangle, A)$ are feasible, then the latter dominates the former whenever $b_j \geq b_i$. Since there is exactly one job between J_i and J_j we call such rules 1-gap rules and refer to them as D_{11} rules where the first index indicates that we interchanged single jobs only.

Similarly, we can derive dominance rules in which more jobs than one are kept between jobs J_i and J_j. Moreover not only single jobs, but also pairs of jobs or triples of jobs or longer tuples of jobs can be interchanged. Now it should be clear that when we refer to a $D_{\alpha\beta}$-rule, we mean a dominance rule in which α-tuples of jobs are interchanged and a string of β jobs is kept inbetween them.

5.5 Initialization

All variants of the proposed procedure are initialized by specifying

- an initial permutation schedule(incumbent solution) and its blocking position;
- an initial upper bound on the optimal value;
- an initial list of active subproblems.

We define these initial objects as follows. As initial incumbent solution we take the schedule corresponding to the SPT sequence $\langle J_1, J_2, \cdots, J_n \rangle$. Obviously, its blocking position k is given by $k = |\{i | b_i < a\}| + 1$. As initial upper bound we take the value of the objective function at the initial incumbent solution. As members of the initial list of active subproblems we take all feasible subproblems among $P^1(\Lambda, \Lambda), P^2(\Lambda, \Lambda), \cdots, P^n(\Lambda, \Lambda)$ whose lower bounds are less than the initial upper bound. The initial list is then obtained by taking nondecreasing order of their lower bounds. Thus initialization involves computation of lower bounds for the subproblems $P^k(\Lambda, \Lambda)$. If the simple bounds described in section 5.2 are used, then we also keep track of the corresponding (possibly infeasible) schedules.

5.6 Termination

The algorithm terminates either by delivering an optimal solution or by exceeding prescribed maximal time. The former case is indicated by the emptiness of the active list. The incumbent of permutation schedule is then optimal and the incumbent upper bound is the optimal value.

Chapter 6

Computational experiments

We have tested the performance of several variants of the proposed branch-and-cut technique using instances where the processing times on the second machine were generated randomly. The objective of these experiments is to evaluate the performance of the developed heuristics. Here we report the results where:

- number of jobs ranges from 9 to 13;
- processing time on the first machine ranges from 31 to 59;
- processing times b_1 and b_n on the second machine are fixed at $b_1 = 30$ and $b_n = 59$, respectively, and remaining processing times on the second machine are generated from a discrete uniform distribution with a range 30 to 59.

We have tested the variants of the proposed lowerbounds and dominance rules. As described in section 5.2, we have proposed three kinds of lowerbounds, in short , simple(B), improved(I) and LP(L) lowerbounds. Moreover in section 5.4, we have designed a group of dominance rule(D_{ij}).

For ease of reference to a particular algorithm, we use the following notation:

- A^B, A^I and A^L denote the algorithms which are based only on the simple bounds, improved bounds and LP-bounds, respectively. No dominance rule is used.
- $A^{\alpha}_{\beta\gamma}$ where $\alpha \in \{B, I, L\}$, $1 \leq \beta \leq 4$ and $0 \leq \gamma \leq 3$, denotes the algorithm based on bounds α and dominance rules D_{ij} with $1 \leq i \leq \beta$ and $0 \leq j \leq \gamma$.

Our experiments consist of five parts. The purpose of each experiment are as follows:

1. the evaluation of the influence of processing time a on the first machine on the difficulty of problem $F2|p_{1j} = a, nmit|\sum C_j$ (see Figure 6.1, 6.2);
    - We show the result of algorithm A^B only. The results of A^I and A^L are similar. We adopt the most simple algorithm such that we have developed.
2. the comparison of efficiency of the lowerbounds algorithm(see Figure 6.3, 6.4);
    - We show the result of algorithm $A^{\alpha}_{\beta 2}$ ($\alpha = B, I, L$ and $1 \leq \beta \leq 3$) only. The results of $A^{\alpha}_{\beta\gamma}$ ($\gamma = 0, 1, 3, 4$) are similar and 2-gap dominance rule is the most effective gap rule.

3. the evaluation of the zero-gap rules D_{i0}(see Figure 6.5, 6.6);

   - We show the result of algorithm $A^I_{\beta 0}$ $(0 \leq \beta \leq 4)$ only. The results of $A^B_{\beta\gamma}$ and $A^L_{\beta\gamma}$ $(1 \leq \gamma)$ are similar and the algorithm with improved bounds is the fastest.

4. the evaluation of nonzero-gap rules D_{ij}(see Figure 6.7, 6.8);

   - We show the result of algorithm $A^I_{\beta\gamma}$ $(0 \leq \beta, \gamma \leq 3)$ only. The results of $A^B_{\beta\gamma}$ and $A^L_{\beta\gamma}$ are similar and the algorithm with improved bounds is the fastest too.

5. Finally, in order to make it possible to compare the proposed technique with various mixed integer solvers, we formulated the problem as the mixed integer linear programming problem. We use the mixed integer formulation of the problem given in the subsection 5.2.3. The results of comparison with the mixed integer solver of the CPLEX system are summarized in Figure 6.9.

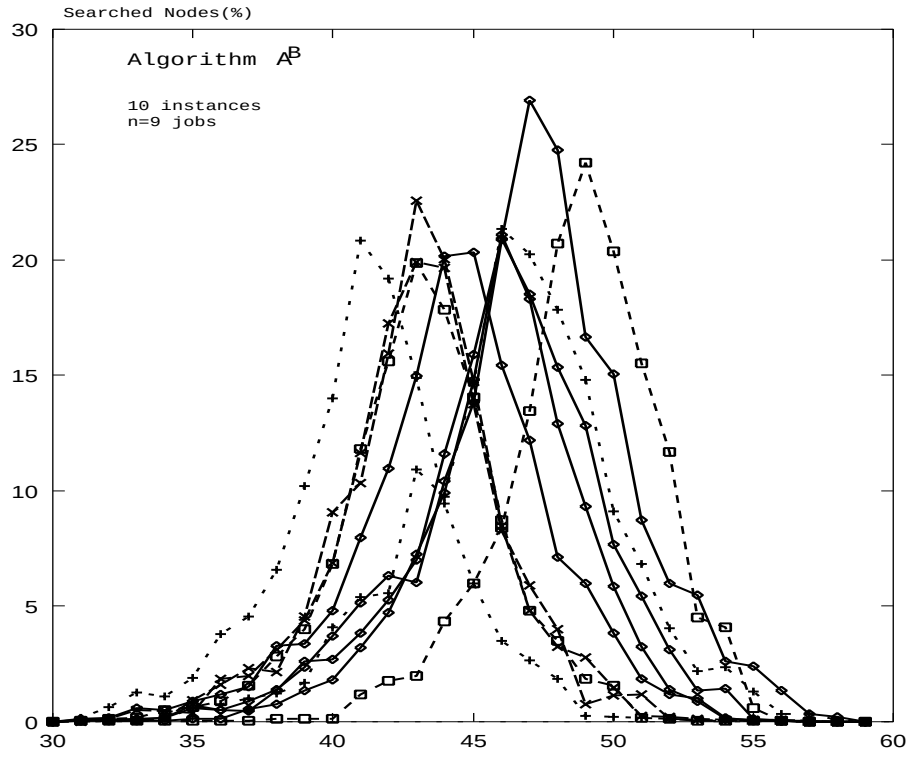

Figure 6.1:

In all variants we used the least-bound strategy for selecting a node from the active list. The percentage of searched nodes refers to the value of $100m/n!$ where m denotes the number of searched nodes and n is the number of jobs. The experiments were done on Sparc Station 20 with 128Mbyte memory.

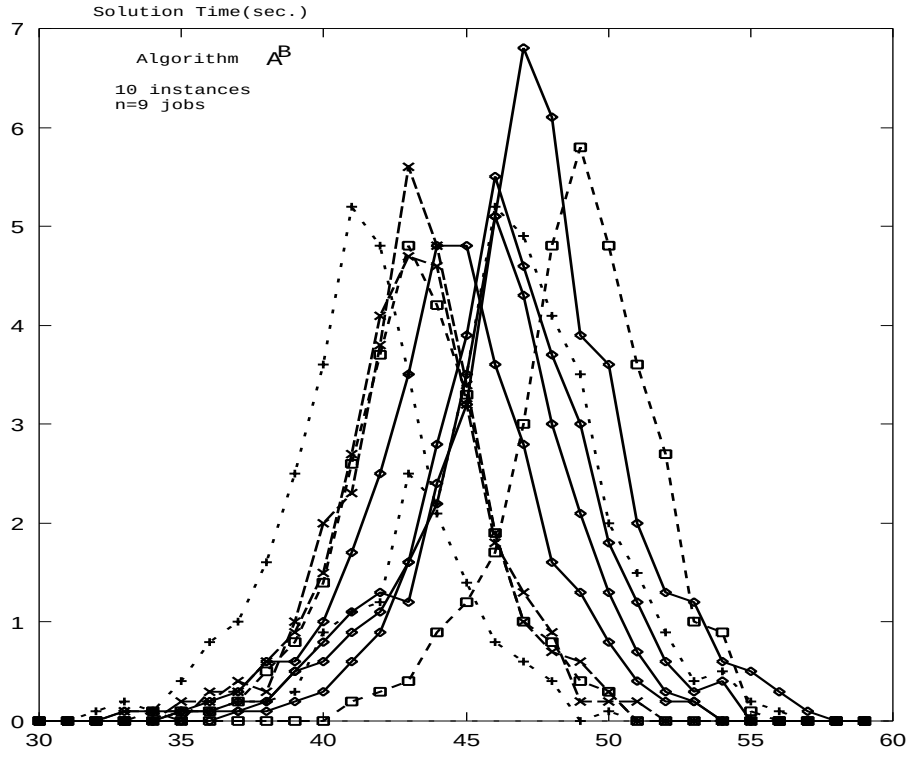

Figure 6.2:

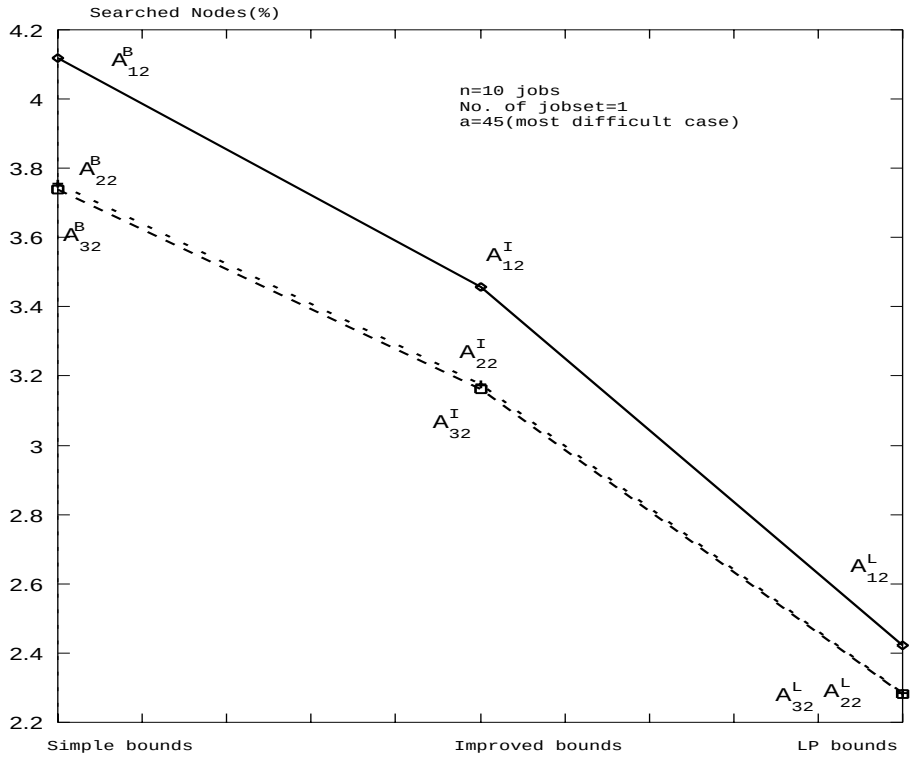

Figure 6.3:

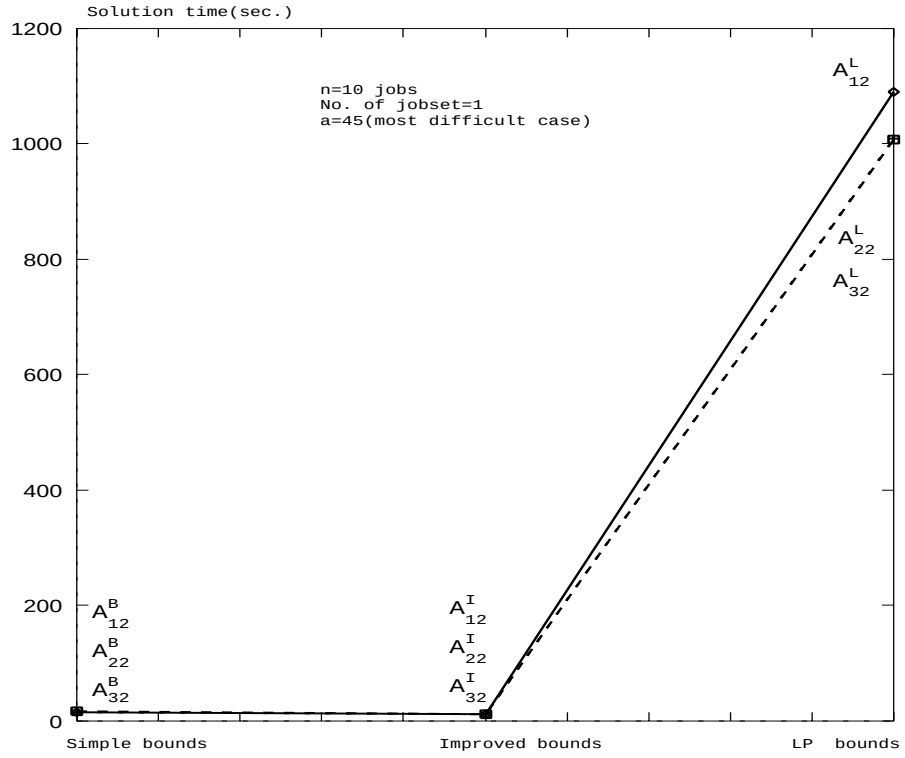

Figure 6.4:

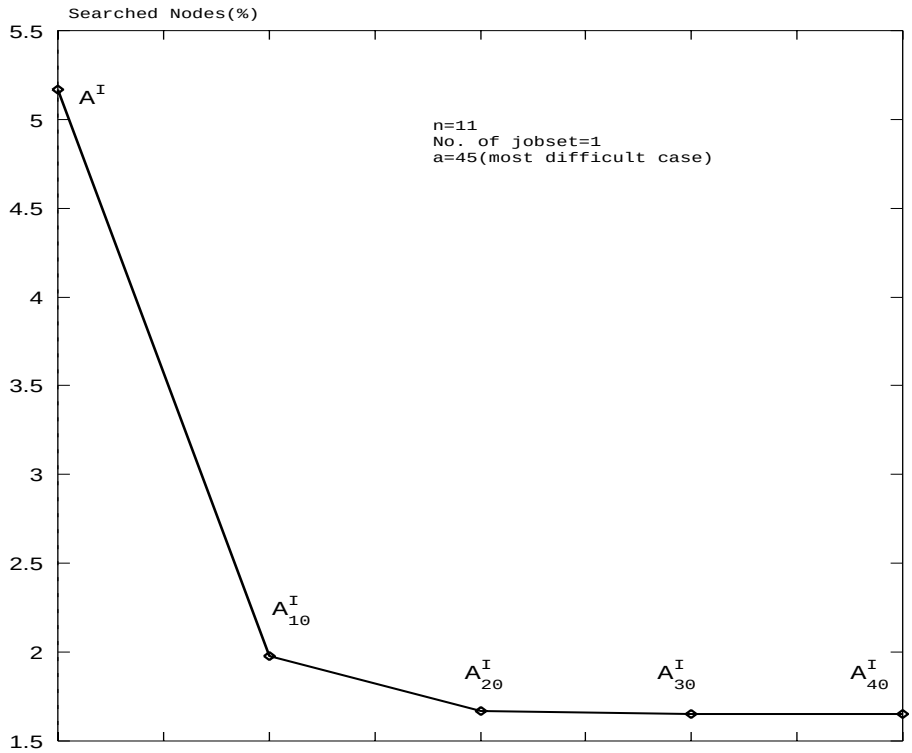

Figure 6.5:

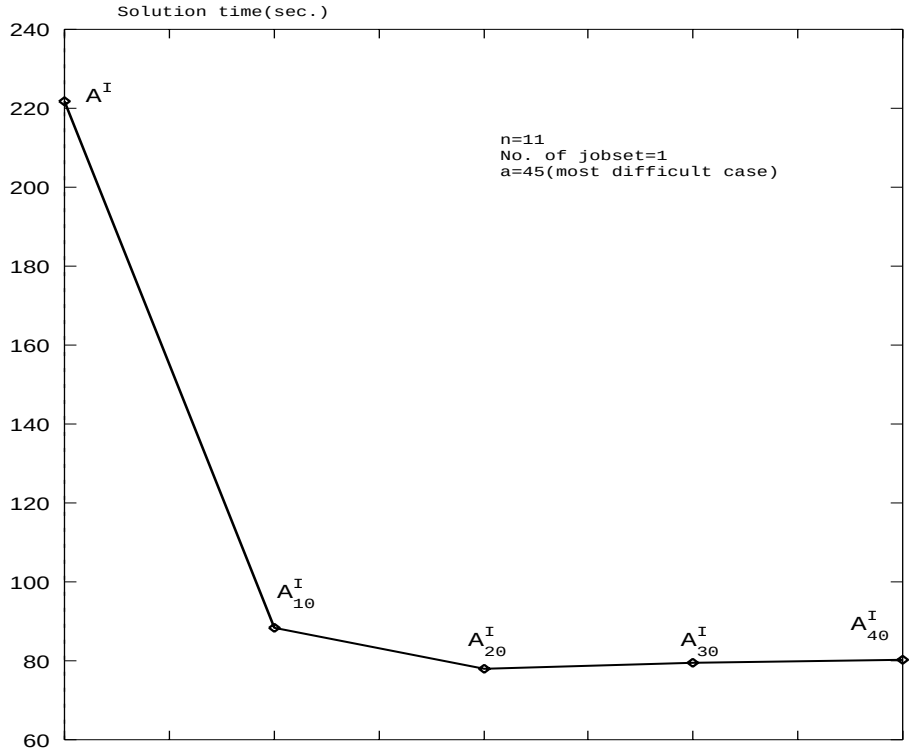

Figure 6.6:

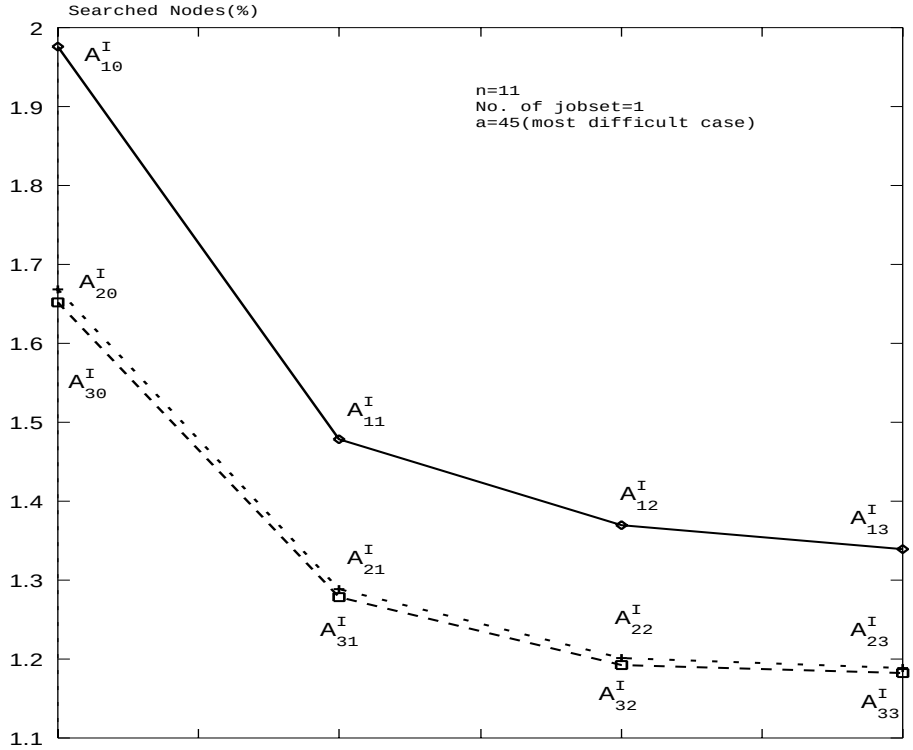

Figure 6.7:

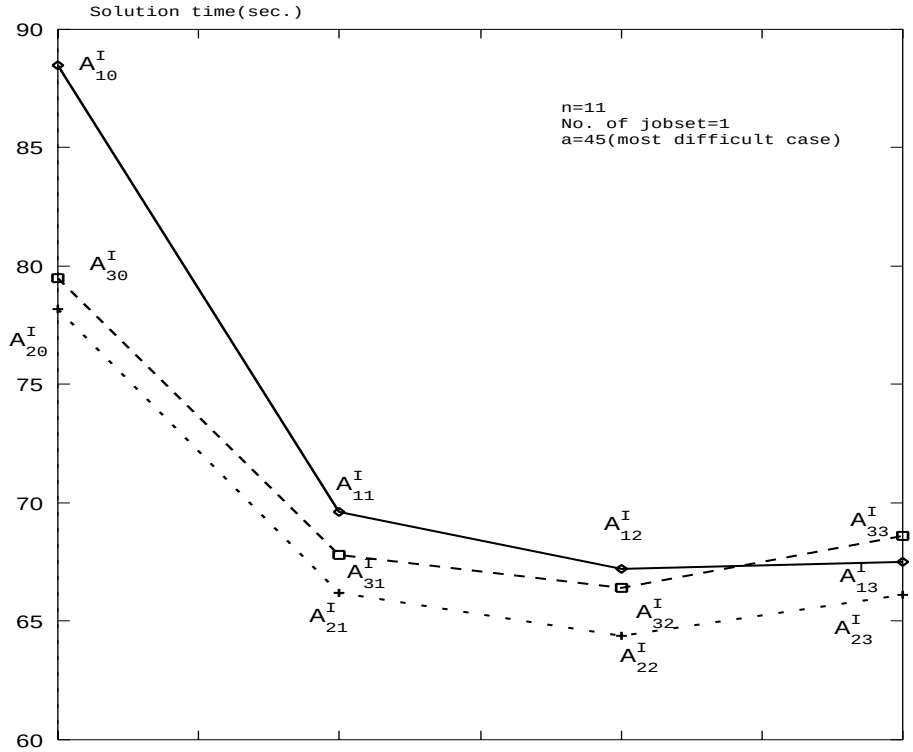

Figure 6.8:

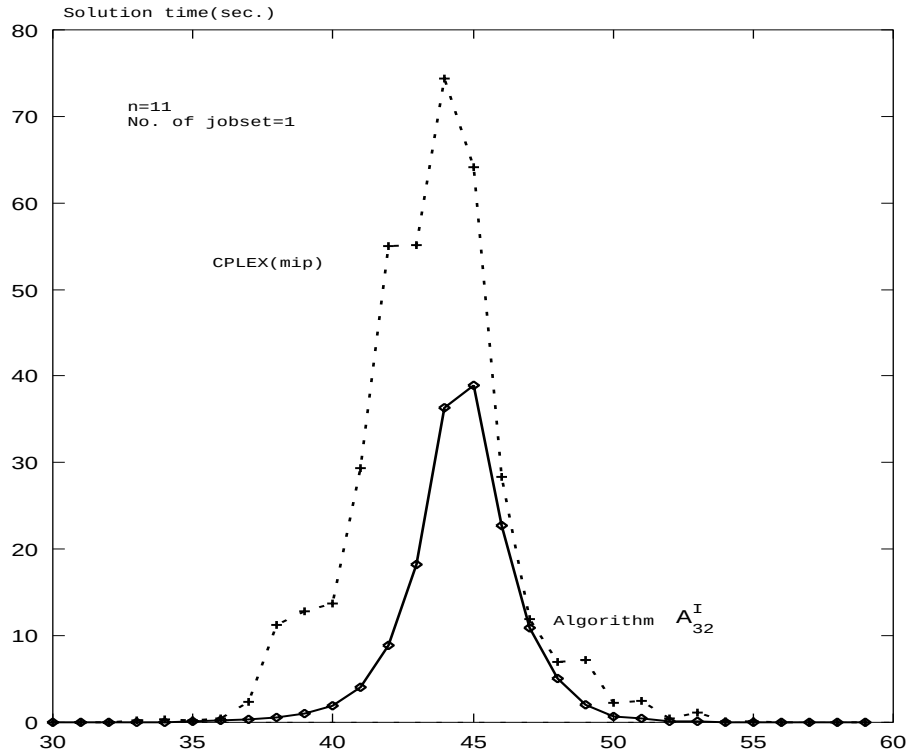

Figure 6.9:

Chapter 7

Concluding remarks

Throughout this dissertation we have been concerned with deterministic flowshop problems where the objective is to minimize the sum of completion times of all jobs. Since the problem is known to be very hard in general, we have restricted our attention to two subcases: problems with special dominance relations among machines and problems with two machines.

In Chapter 3, we have considered flowshop problems with special structured machine dominance relation. Adiri and Pohoryles [1] designed two polynomial time algorithms constructing the best permutation schedules for $Fm|nmit, idm|\sum C_j$ and $Fm|nmit, ddm|\sum C_j$ respectively. We have proved that all of the above algorithms construct not only the best permutation schedules for the given problem, but that they in fact produce optimal schedules. In fact we have proved a much more general result, namely that under the above machine dominance constraints, the search for optimal schedule can be restricted to the set of all permutation schedules not only for the sum of completion times criterion but for an arbitrary regular objective function.

In Chapter 4 and 5, we have considered the case of two machines. Two theorems of paper [1] which deal with the $F2|nmit|\sum C_j$ problem contain incorrect claims. We have showed that both claims are incorrect by constructing a counter example. Moreover we have managed to prove another statement which has a similar "flavor" as the original (incorrect) claims. Furthermore we have described several variants of the branch-and-cut technique for solving the $F2|p_{1j} = a, nmit|\sum C_j$ problem and reported results of computational experiments. Their analysis leads to the following conclusions.

- Figures 6.1 and 6.2 suggest that the instances in which processing time a on the first machine is around the middle of the range of processing times b_i on the second machine are the most difficult. The reason is that when a is around the middle of the range of processing times on the second machine, many candidate jobs which might be assigned either after or before blocking position exist and, therefore, subproblem P^k has many feasible solutions.

- Figure 6.3 shows that the improved bounds and LP bounds decrease the number of searched nodes. Figure 6.4 shows that the improved bounds decrease solution times too. In the more detailed observation, we have realized that LP bounds is the most effective around the root of tree.

- Figure 6.5 and 6.6 show that D_{10}-rules and D_{20}-rules increase the efficiency of algorithms, but there is no improvement when dominance is based on the interchange

of i-tuples for $i > 2$.

- Similarly figures 6.7 and 6.8 show that D_{i1}-rules and D_{i2}-rules are effective in our algorithms, but no improvement is achieved when gap is based on more than 2 jobs. Moreover these figures imply that D_{ij}-rules ($i = 1, 2$ and $0 \leq j \leq 2$) are the most effective around the root of the tree.

- A comparison with the mixed integer solver of CPLEX indicates that the proposed branch-and-cut technique is faster than CPLEX (see Figure 6.9).

Bibliography

[1] Adiri, I. and Pohoryles, D. *Flowshop/No-idle or No-wait scheduling to minimize the sum of completion times*, Naval Research Logistics Quartely 29, 495-504 (1982)

[2] Baker, K. R. *Introduction to sequencing and scheduling*, John Wiley & Sons, (1974)

[3] Conway, R. W., Maxwell, W. L., and Miller, L. W. *Theory of Scheduling*, Addison-Wesley, Reading, Mass. (1967)

[4] Della Croce, F., Narayan, V., and Tadei, R. *The two-machine total completion time flow shop problem.* , European Journal of Operation Research, 90, 227-237(1996)

[5] Garey, M. R. , Johnson, D. S., and Sethi, R. *The complexity of flowshop and jobshop scheduling*, Mathematics of Operations Research 1, 117-129 (1976)

[6] Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kan, A. H. G. *Optimization and approximation in deterministic sequencing and scheduling*, a survey. Annals of Discrete Mathematics 5, 287-326 (1979)

[7] Ho, J. C. and Gupta, J. N. D. *Flowshop Scheduling with Dominant machines*, Computers Ops. Res., 22, No. 2, 237-246(1995)

[8] Hoogeveen, J. A., and Kawaguchi, T. *Minimizing total completion time in a two-machine flowshop: analysis of special cases.*

[9] Ignall, E. and Schrage, L. *Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems*, Oper. Res., 13, 400-412 (1965)

[10] Johnson, S. M. *Optimal Two-and Three-Stage Production Schedules with Setup Times Included*, Naval Research Quarterly, Vol. 1, No. 1 (March, 1954)

[11] Kohler, W. H., and Steiglitz, K. *Exact, approximate and guaranteed accuracy algorithms for the flowshop problem n/2/F/$\bar{F}$.*, ACM, Vol. 22, No. 1, 106-114(1975)

[12] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B. *Sequencing and Scheduling : Algorithms and Complexity*, Handbooks in OR & MS, Elsevir Science Publishers B. V., Vol. 4 445-522(1993)

[13] Lomnicki, Z. *A Branch-and-Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem*, Operational Research Quarterly, Vol.16, No. 1 (March, 1965)

[14] Monma, C. L. and Rinnooy Kan, A. H. *A Concise Survey of Efficiently Solvable Special Cases of the Permutation Flow-Shop Problem*, RAIRO, Vol. 17, No. 2 105-119(1983)

[15] Van de Velde, S. L. *Minimizing the sum of the job completion times in the two-machine flowshop by Lagrangean relaxation.*, Annals of Operations Research, 26, 257-268(1990)

Publications

[1] M. Okada, K. Tanaka and M. Vlach: "Minimizing total completion time in a flow shop under a no-idle constraint," SYMPOSIUM ON OPERATIONS RESEARCH 1997, Jena(Germany),September 3-5,1997.

[2] M. Okada, K. Tanaka and M. Vlach: "No-Idle Flowshop Problem with Sum of Completion Times Performance Criterion," , 統計数理研究所共同研究リポート 104「最適化:モデリングとアルゴリズム 11」, pp. 27-45, 1997. 12.

[3] 岡田政則, 田中圭介,Milan Vlach: "Minimizing Total Completion Time in the Two-Machine Flow Shop with No Machine Idle Time Constraint," 研究集会「最適化：モデリングとアルゴリズム」, 1997. 3. 27-28.

[4] M. Okada, K. Tanaka and M. Vlach: "TWO MACHINE FLOW-SHOP WITH NO MACHINE IDLE TIME TO MINIMIZE THE SUM OF COMPLETION TIMES", 日本 OR 学会「意志決定と OR」研究集会講演論文集, pp. 21-22, 1996. 11.