SHOP SCHEDULING WITH AVAILABILITY CONSTRAINTS

Mikhail A. Kubzin

A thesis submitted in partial fulfilment of the requirements of the University of Greenwich for the degree of Doctor of Philosophy

June 2005

The University of Greenwich,
School of Computing and Mathematical Science,
Park Row, Greenwich, SE10 9LS

Contents

Acknowledgements

The author wishes to thank Professor Vitaly Strusevich for his continuous guidance, wise and sound advices and good humour throughout. Additionally, he wishes to thank Professor C.N. Potts for providing help and dialogue along the way.

The author is grateful to Professor Colin Reeves for useful comments which allowed to improve the thesis.

Financially, the author would like to recognise the support of the University of Greenwich which made this research possible.

Finally, the last but not the least of all, I owe most deepest and loving thanks to my lovely wife Vita for her encouragement and support.

Abstract

Scheduling Theory studies planning and timetabling of various industrial and human activities and, therefore, is of constant scientific interest. Being a branch of Operational Research, Theory of Scheduling mostly deals with problems of practical interest which can be easily (from a mathematical point of view) solved by full enumeration and at the same time usually require enormous time to be solved optimally. Therefore, one attempts to develop algorithms for finding optimal or near optimal solutions of the problems under consideration in reasonable time. If the output of an algorithm is not always an optimal solution then the worst-case analysis of this algorithm is undertaken in order to estimate either a relative error or an absolute error that holds for any given instance of the problem.

Scheduling problems which are usually considered in the literature assume that the processing facilities are constantly available throughout the planning period. However, in practice, the processing facility, e.g. a machine, a labour, etc., can become non-available due to various reasons, e.g. breakdowns, lunch breaks, holidays, maintenance work, etc. All these facts stimulate research in the area of scheduling with non-availability constraints. This branch of Scheduling Theory has recently received a lot of attention and a considerable number of research papers have been published. This thesis is fully dedicated to scheduling with non-availability constraints under various assumptions on the structure of the processing system and on the types of non-availability intervals.

Chapter 1

Introduction

Scheduling is a form of decision-making which plays an important role in many areas such as manufacturing, transport, computing, etc. Theory of Scheduling studies mathematical models that arise in the planning and timetabling of various activities. While informally scheduling has been used for centuries it began to be taken seriously at the beginning of the previous century with the work of Gantt. However, it took the first scheduling publications almost half a century to appear in the Operational Research literature. First scheduling algorithms were formulated and published by Johnson [68], Smith [137] and Jackson [67]. Traditionally, scheduling problems are formulated in terms of processing jobs on machines. A scheduling problem is to find a feasible schedule that satisfies all processing requirements and optimises a certain objective function, which usually depends on jobs' completion times.

In the early seventies due to famous works of Cook [36] and Karp [70] on computational complexity, research focused mainly on complexity aspects of scheduling problems. Nowadays it is commonly accepted to call a problem to be *easy* if it admits an algorithm such that it finds an optimal solution and its running time is bounded by a polynomial in the size of the input data (this problem is said to belong to the class of P-hard problems). If for some optimisation problem the existence of such an algorithm is highly unlikely then this problem is assumed to be *hard* (this problem is said to belong to

the class of NP-hard problems). The crucial feature of scheduling theory is the fact that it has a virtually unbounded number of problem types, see, e.g., Conway et al. [35], Baker [9], Coffman [34], Rinnooy Kan [119], Lenstra [96], French [41], Tanaev et al. [142]. Most scheduling problems of practical interest belong to the class of NP-hard problems.

There are two ways of solving NP-hard problems. First, we can try to find an optimal solution but this requires a time consuming search and sometimes the running time of such algorithms may appear to be unacceptable. Second, we can search not for an optimal solution but for a feasible solution which is close enough to the optimum. Certainly, if we select the latter way we wish to obtain a heuristic solution which is as close to the optimum as possible and spend on finding such a solution as less time as possible.

Most of the literature on scheduling studies scheduling problems under the assumption that all machines are continuously available for processing the jobs. This, however, is not always true in practice. Thus, it is worth studying such problems with machine *availability constraints*. In many practical situations the processing machines may not be continuously available throughout the planning period due to maintenance requirements or rest periods which have to be taken into account. Scheduling problems with machine availability constraints have been extensively studied since the 1990s, see [91, 130, 131] for surveys in this area.

There are two major types of sheduling problems considered in the literature: one-stage models and multi-stage (or shop scheduling) models. Let us recall that in shop scheduling a decision-maker is given a set of jobs and a set of machines. All jobs have to be processed on given machines. The sequence of machines in which a job undergo processing in the system is called a *processing route* of this job. If all jobs have the same processing route than this processing system is called a *flow shop*. If processing routes are not fixed and can be defined for each job by the decision-maker then this processing system is called an *open shop*.

A polynomial-time algorithm that outputs a heuristic solution that is $\rho \geq 1$ times worse the optimal solution is called a $\rho-$*approximation algorithm.* A family of $(1+\varepsilon)-$approximation algorithms (where $\varepsilon > 0$) is called a *polynomial-time approximation scheme,* or a *PTAS,* if the running time is polynomial in the length of the problem input. If additionally the running time of a PTAS is polynomial with respect to $1/\varepsilon$, then it is called a *fully polynomial-time approximation scheme*, or an FPTAS. We will give formal definitions of approximation algorithms, PTAS and FPTAS in Section 1.2.

The main aim of this work is to design fast polynomial-time heuristic algorithms for shop scheduling problems with availability constraints and to prove that they guarantee to output solutions close to the optimum solutions even in the worst case.

The layout of the rest of this thesis is as follows. Chapter 1 provides a general background of the theory of scheduling. Here, we define terminology and notation that is used throughout the thesis and introduce some basic scheduling models. Furthermore, we make an overview of the present achievements in Scheduling Theory and briefly describe some basic algorithms which will be used later.

In Chapter 2 we introduce specific terminology and review the existing literature on scheduling with availability constraints.

Chapters 3 through 6 are devoted to our contribution to the field of scheduling with availability constraints. Each chapter addresses a particular shop scheduling problem. For all problems we consider in this thesis, the objective function is the maximal completion time, i.e., the makespan.

In Chapter 3 we consider the two-machine flow shop scheduling problem with non-availability intervals either on the first machine or on the second machine under various scenarios which are described in Chapter 2. We start with presenting a dynamic programming algorithm for the resumable scenario and several non-availability intervals on one of the machines. Then we

demonstrate how to convert the available dynamic programming algorithms to FPTAS's. Since the running time of these FPTAS's is fairly large, a fast heuristic algorithm with a guaranteed worst-case ratio of 3/2 is presented for the problem with holes on the first machine. Finally we describe a PTAS for the two-machine flow shop problem with a single non-availability interval on one of the machines under the semi-resumable scenario.

In Chapter 4 we analyze the two-machine flow shop problem with no-wait in process with one non-availability interval on one of the machines under various scenarios. We show that our problem is NP-hard irrespective of the scenario. Then we present a 3/2-approximation algorithm that is applicable to any scenario and a 4/3-approximation algorithm for the resumable scenario.

In Chapter 5 we present a PTAS for the open shop problem with several non-availability intervals on one of the two machines and a PTAS for the two-machine open shop with a single non-availability interval on each machine.

In Chapter 6 various scheduling problems subject to machine preventive maintenance are considered. For the two-machine open shop problem we present a polynomial-time algorithm, for the two-machine flow shop problem we prove that this problem becomes NP-hard even if the length of the maintenance interval depends linearly on its starting time. We also give a pseudopolynomial dynamic programming algorithm and two approximation algorithms, including an FPTAS. For the two-machine flow shop problem with no-wait in process subject to a single maintenance interval we present a PTAS.

Finally in Chapter 7 we give some concluding remarks and summarize the obtained results.

1.1 Shop Scheduling

In this section we provide basic notation and recall some important definitions related to shop scheduling.

In a general shop scheduling model, we are given a set $\mathcal{M}=\{M_1, M_2, \ldots, M_m\}$ of machines and a set $N = \{J_1, J_2, \ldots, J_n\}$ of jobs to be processed on these machines. The processing of each job $J_j \in N$ consists of r_j stages. Each job $J_j \in N$ on each stage q, $q = 1, 2, \ldots, r_j$, can be processed on one machine L from the predefined subset of machines $\mathcal{M}_q^i \subseteq \mathcal{M}$. Throughout the thesis we assume that $\left|\mathcal{M}_q^i\right| = 1$, i.e., in each stage each job can be processed on only one machine. It is assumed that each machine can process at most one job at a time and each job can be processed at most on one machine at a time.

The *processing time* of job J_j on machine M_i is denoted by p_{ij}. If either the processing time of job J_j does not depend on the machine or the job has to be processed only on one given machine we omit the subscript i.

The *workload* of machine M_i is the total processing time of all jobs which have to be processed on this machine.

The earliest time at which job J_j is ready for processing in the system is called the *release date* of this job. The release date is denoted by r_j.

The *due date* d_j of job J_j represents the committed completion time. In fact the completion of the job after the due date is allowed but a penalty usually occurs.

If the due date must be met then we will refer to it as a *deadline* and denote by $\bar{d}_j$.

Sometimes it is necessary to take into account a job priority. For this purpose we consider *weight* w_j of job J_j which determines the importance of this job relatively to other jobs in the system.

In *parallel machine* scheduling there are m identical machines in the processing system and we are given a set of jobs $N = \{J_1, J_2, \ldots, J_n\}$. Each

job J_j requires a single operation O_j with processing time p_j and can be processed on any of the m machines.

In *flow shop* scheduling model we are given a set of jobs $N = \{J_1, J_2, \ldots, J_n\}$ and m machines M_i, $i = 1, 2, \ldots, m$. Each job $J_j \in N$ consists of the set of operations $\{O_{1j}, \ldots, O_{mj}\}$, where O_{ij} denotes an operation of job J_j on machine M_i, the processing time of each operation O_{ij} is known in advance and is denoted by p_{ij}. All jobs are assigned to the same processing route $(M_1, M_2, \ldots, M_m)$ of machines.

If the order in which each machine processes the jobs is identical for all machines then this schedule is called a *permutation* schedule. Such a schedule may be specified just by a permutation of job indices.

An important variant of the flow shop problem is the so-called flow shop with the *no-wait* in process constraint. In the no-wait environment each job once started has to be processed on all machines without interruptions and inter-stage delays until it is completed.

The classical scheduling model which is known as an *open shop* was introduced by Gonzalez and Sahni [52]. In the open shop problem, each job has to be processed on each machine, but the machine routes for each job are not specified in advance and have to be chosen. Formally, there are m machines M_i, $i = 1, 2, \ldots, m$, in the system and the set of jobs $N = \{J_1, J_2, \ldots, J_n\}$. Each job $J_j \in N$ can be viewed as the set of operations $\{O_{1j}, O_{2j} \ldots, O_{mj}\}$, where O_{ij} denotes an operation of job J_j on machine M_i, the processing time of each operation O_{ij} is known in advance and is denoted by p_{ij}. The order in which operations undergo the processing is part of the decision-making, different jobs being allowed to have different routes.

1.2 Problem Classification

Since the number of scheduling problems seems to be virtually unbounded an effective system of problems classification is required. Here we use the

commonly accepted three-field classification scheme introduced by Graham et al. [57]. According to this scheme a scheduling problem is coded by a string that consists of three main parts: the processing system, extra conditions and the objective function. Each scheduling problem can be described using the three-field notation $\alpha|\beta|\gamma$ such that α represents the machine environment, β defines the processing conditions, and γ is the objective function which is used for evaluating feasible schedules. Below we give a detailed description of this notation system since it will be intensively used throughout the thesis.

Further, we assume that $\circ$ denotes the empty symbol.

1.2.1 Machine Environment

This field defines the configuration of the processing system. This field has the form $\alpha = \alpha_1\alpha_2$, where α_1 and α_2 are interpreted as follows.

- $\alpha_1 \in \{\circ, P, F, O, J, V\}$:

  i. $\alpha_1 = \circ$: single machine;

  ii. $\alpha_1 = P$: identical parallel machines;

  iii. $\alpha_1 = O$: an open shop;

  iv. $\alpha_1 = F$: a flow shop;

  v. $\alpha_1 = J$: a job shop.

- $\alpha_2 \in \{\circ, m\}$:

  i. $\alpha_2 = \circ$: the number of machines/stages is arbitrary;

  ii. $\alpha_2 = m$: there is a fixed number m of machines.

1.2.2 Job Characteristics

For each job J_i we are given its processing times on all machines which are non-negative integers. Also each job may be characterized by its availability for processing and a due date, its dependence on other jobs, the possibility of interruptions in the processing, etc. All this information is included in the second field of the three-field notation.

The second field $\beta \subseteq \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$ indicates job characteristics as follows.

- $\beta_1 \in \{\circ, p_{ij} = 1, p_{ij} \in \{0, 1\}\}$:

  i. $\beta_1 = \circ$: processing times are arbitrary;

  ii. $\beta_1 = p_{ij} = 1$: each operation has unit processing time;

  iii $\beta_1 = p_{ij} \in \{0, 1\}$: each operation has either unit or zero processing time.

- $\beta_2 \in \{\circ, r_j\}$:

  i. $\beta_2 = \circ$: no release dates are specified;

  ii. $\beta_2 = r_j$: jobs have release dates.

- $\beta_3 \in \{\circ, d_j, \overline{d_j}\}$:

  i. $\beta_3 = \circ$: no due dates/deadlines are specified;

  ii. $\beta_3 = d_j$: jobs have due dates;

  iii. $\beta_3 = \overline{d_j}$: jobs have deadlines.

- $\beta_4 \in \{\circ, pmtn\}$:

  i. $\beta_4 = \circ$: no preemption is allowed;

  ii. $\beta_4 = pmtn$: operations of jobs may be preempted.

- $\beta_5 \in \{\circ, \textit{no-wait}\}$:

  i. $\beta_5 = \circ$: the no-wait requirement does not apply;

  ii. $\beta_5 = \textit{no-wait}$: there are the "no-wait in process" restrictions.

1.2.3 Objective Function

The third field γ specifies the objective function to be minimised. The objective function is a penalty function which depends on the completion times of the jobs. Given a schedule S, we can compute for job J_j:

- the *completion time* $C_j(S)$;
- the *lateness* $L_j(S) = C_j(S) - d_j$;
- the *earliness* $E_j(S) = \max\{d_j - C_j(S), 0\}$;
- the *tardiness* $T_j(S) = \max\{C_j(S) - d_j, 0\}$.

Moreover, if f_j is a non-decreasing cost function, then the cost of completion time of job j is $f_j(S) = f_j(C_j(S))$. If no ambiguity arises regarding the schedule under consideration, we may drop the reference to a particular schedule and write C_j, L_j, E_j, T_j, and f_j, respectively.

Some commonly used optimality criteria involve the minimisation of:

- the maximum completion time, i.e. *makespan*, $C_{\max} = \max_j C_j$;
- the *maximum lateness* $L_{\max} = \max_{j \in N} L_j$;
- the maximum cost $f_{\max} = \max_{j \in N} f_j$;
- the *total (weighted) completion time* $\sum_{j \in N} (w_j) C_j$;
- the *total (weighted) tardiness* $\sum_{j \in N} (w_j) T_j$;
- the *total (weighted) earliness* $\sum_{j \in N} (w_j) E_j$;

- the *total cost* $\sum_{j \in N} f_j$.

To summarize, the third field γ defines the optimality criterion, which involves the minimisation of

$$\gamma \in \{C_{\max},\ L_{\max},\ f_{\max}, \sum(w_j)C_j,\ \sum(w_j)T_j,\ \sum(w_j)E_j,\ \sum f_j\}.$$

It should be noted that some situations may require more than one of these criteria to be considered.

The results discussed in this thesis are primarily devoted to the problems of minimising the function $C_{\max}(S)$. A schedule minimising the makespan is called *time-optimal*.

1.2.4 More Notation and Definitions

We recall some common scheduling dispatching rules. We will say that the jobs obey the Earliest Due Date rule (EDD) if they are sequenced in non-decreasing order of their due dates. Analogously, if the jobs are sequenced in non-increasing (non-decreasing) order of their processing times we will say that these jobs follow Longest (Shortest) Processing Time rule and denote this tule by LPT (SPT).

If $|\mathcal{M}| = 2$, i.e., there are only two processing machines, we will denote them by the letters A and B. The processing times of job J_j on machines A and B we will denote by a_j and b_j, respectively. Define

$$a(Q) = \sum_{j \in Q} a_j,\ b(Q) = \sum_{j \in Q} b_j$$

for a non-empty set $Q \subseteq N$ of jobs, and define $a(\emptyset) = b(\emptyset) = 0$.

For a schedule S, let $R_{jL}(S)$ and $C_{jL}(S)$ denote the starting time and the completion time, respectively, of operation O_{jL}, $J_j \in N, L \in \{A, B\}$. Let $C_A(S)$ or $C_B(S)$ denote the time that machine A or respectively B completes all its jobs in schedule S.

Further we assume that the reader is familiar with Computational Complexity and classes P and NP, otherwise a detailed discussion can be found in [45].

Let us recall that a polynomial-time algorithm for a minimization problem that creates a schedule with the makespan that is at most $\rho \geq 1$ times the optimal value is called a $\rho-$*approximation* algorithm; the value of ρ is called a *worst-case ratio bound.* If a problem admits a $\rho-$approximation algorithm it is said to be *approximable within a factor* ρ. A worst-case ratio bound is called *tight* if for any given $\varepsilon > 0$ there exists an instance of the problem for which the apprximation algorithm delivers a heuristic solution $(\rho - \varepsilon)$ times the optimal value.

A family of $\rho-$approximation algorithms is called a *polynomial-time approximation scheme,* or a *PTAS,* if $\rho = 1 + \varepsilon$ for any fixed $\varepsilon > 0$ and the running time is polynomial in the length of the problem input. If additionally the running time of a PTAS is polynomial with respect to $1/\varepsilon$, then it is called a *fully polynomial-time approximation scheme,* or an *FPTAS.* We recall that, unless P=NP, there does not exist an FPTAS for any strongly NP-hard problem.

Further, we use the following well-known NP-complete problem in the proofs of NP-hardness of certain problems.

PARTITION. Given r positive integers $e_i, i \in R = \{1, 2, \ldots, r\}$, and an integer E such that $\sum_{i \in R} e_i = 2E$, does there exist a partition of set R into two subsets R_1 and R_2 such that $\sum_{i \in R_1} e_i = \sum_{i \in R_2} e_i = E$?

1.3 Flow Shop Problem

Let us recall that in flow shop scheduling model we are given a set of jobs $N = \{J_1, J_2, \ldots, J_n\}$ and m machines M_i, $i = 1, 2, \ldots, m$. Each job $J_j \in N$ consists of the set of operations $\{O_{1j}, \ldots, O_{mj}\}$, where O_{ij} denotes an operation of job J_j on machine M_i, the processing time of each operation O_{ij}

is known in advance and is denoted by p_{ij}. All jobs are assigned the same processing route $(M_1, M_2, \ldots, M_m)$ of machines.

In one of the first papers on deterministic machine scheduling, Johnson gives an $O(n \log n)$-time algorithm for finding an optimal permutation schedule for problem $F2||C_{\max}$, see [68] and Section 1.4.1. Garey et al. [44] prove that problem $F3||C_{\max}$ is strongly NP-hard which yields that more general problems $Fm||C_{\max}$ and $F||C_{\max}$ are strongly NP-hard as well.

Most research on the flow shop problems has focused on permutation schedules. Conway et al. [35] show that for $Fm||C_{\max}$ there always exists an optimal schedule with the same job ordering on the first two machines M_1 and M_2 and with the same job ordering on the last two machines M_{m-1} and M_m. Thus, for both $m = 2$ and $m = 3$ an optimal schedule for problem $Fm||C_{\max}$ can always be found in the class of permutation schedules. However, for $m \geq 4$ that is not true. Potts et al. [125] analyze the worst-case performance of permutation schedules for the flow shop problems with more than 3 machines. They present a family of $Fm||C_{\max}$ problems such that the worst-case performance ratio of the permutation schedules with respect to the global optimum is not less than $\frac{1}{2}\lceil\sqrt{m}+\frac{1}{2}\rceil$ and, therefore, is not bounded by any finite constant.

Gonzalez and Sahni [52] prove strong NP-hardness of problem $F3|pmtn|C_{\max}$. Neumytov and Sevastianov [106] study a special case of the three-machine flow shop in which each job consists of only two operations, one of them (the last operation for all jobs, or by symmetry, the first operation for all jobs) has to be performed on a particular machine, the same for all jobs. It is shown that even such a restricted variant of $F3||C_{\max}$ is strongly NP-hard. Lenstra et al. [97] show that problems $F2|r_j|C_{\max}$ and $F2||L_{\max}$ are strongly NP-hard. Cho and Sahni [33] consider problems $F2|r_j, pmtn|C_{\max}$, $F2|pmtn|L_{\max}$ and prove that they are NP-hard in the strong sense. Garey et al. [44] show that problem $F2||\sum C_j$ is strongly NP-hard while Du and Leung [37] prove that problem $F2|pmtn|\sum C_j$ is strongly NP-hard too.

Gonzalez and Sahni [53] introduce the concept of busy schedules. A flow shop schedule is called *busy*, if at any time during the interval $[0, C_{\max}]$ there is at least one machine processing a job. For problem $F||C_{\max}$ they show that for any busy schedule S_B the following bound

$$\frac{C_{\max}(S_B)}{C_{\max}(S^*)} \leq m$$

holds and is tight. For problem $Fm||C_{\max}$ it is clear that sequencing the jobs in an arbitrary order yields a trivial m-approximation algorithm. This bound was improved by Röck and Schmidt [121]. They present a polynomial time $\lceil m/2 \rceil$-approximation algorithm. See [102, 142] for further discussion of problem $Fm||C_{\max}$. Later, Chen et al. [24] propose a heuristic algorithm for this problem with the worst-case performance ratio of $\frac{m}{2}$ if m is even and $\frac{m}{2} + \frac{1}{6}$ if m is odd. Probably the most notable theoretical achievement in the flow shop approximation, is a PTAS for problem $Fm||C_{\max}$ by Hall [61].

Nowicki and Smutnicki study the permutation flow shop problem and present some approximation methods with the tight worst-case performance ratio of $\lceil m/2 \rceil$ and $m/\sqrt{2} + O(1/m)$, see [108, 109, 110, 111] for details.

For problem $F||C_{\max}$ heuristic algorithms that require reasonable computational effort are not known to provide even a constant worst-case performance ratio ρ. Williamson et al. [150] prove that there does not exist an approximation algorithm with a worst-case ratio less than 5/4, unless P=NP. Shmoys et al. [136], Goldberg et al. [50] and Feige and Scheideler [40] present polynomial time approximation algorithms with the worst-case ratio of $O(\log m \log \log m)$.

For $F3||C_{\max}$ problem, Chen et al. [24] give an $O(n \log n)-$time algorithm with a worst-case ratio $\rho = 5/3$. The same worst-case ratio is achievable in $O(n^3 \log n)$ time for problem $F2|r_j|C_{\max}$, see [124].

Hall [60] considers $F2|r_j|C_{\max}$ problem and shows that it admits a polynomial time approximation scheme. Kovalyov and Werner [74] present another PTAS for $F2|r_j|C_{\max}$ problem, which improves the one due to Hall in terms

of computational efficiency.

For $F2||C_{\max}$ problem, by establishing lower bounds in the algebraic computation tree model, Rote and Woeginger [120] show that the time complexity of Johnson's algorithm is in fact the best possible; they also develop a fully-polynomial time approximation scheme for this problem which requires $O(n \log \frac{1}{\varepsilon})$ time for any $\varepsilon > 0$.

Hoogeveen et al. [65] prove that $F||\sum C_j$ problem belongs to the class of APX-hard (or Max SNP-hard) problems, and, therefore, does not possess a PTAS, unless P=NP, see Section 1.9 for the discussion of non-approximability and the description of the class of APX-hard problems.

1.4 Flow Shop Algorithms

In this section we consider in details some results for the two-machine flow shop problem which will be used further in this thesis. Recall that we are given two machines A and B and the set of jobs N to be processed on these two machines. All jobs have to follow the same route (A, B) of processing. Johnson [68] shows that there exists an optimal solution for this problem in the class of permutation schedules.

Let π be some permutation of indices, $\pi(j)$ be the j-th element in this permutation and S_π be the schedule associated with this permutation. Such schedule S_π can be constructed by a given permutation π in the following way. We schedule the jobs on each machines according to this permutation and make each operation to start as early as possible. Given a permutation π of jobs, the makespan of schedule S_π can be then defined as

$$C_{\max}(S_\pi) = \max_{1 \le \mu \le n} \left\{ \sum_{j=1}^{\mu} a_{\pi(j)} + \sum_{j=\mu}^{n} b_{\pi(j)} \right\}. \tag{1.1}$$

If $\mu = u$ is an index which in fact delivers the maximum in (1.1) then we refer to job $J_{\pi(u)}$ as a *critical* job. Thus, a critical job starts its processing on machine B as soon as it completes its processing on machine A without

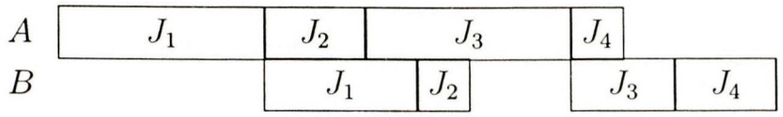

Figure 1.1: Schedule S_π

any delay. The makespan of a schedule that contains a critical job $J_{\pi(u)}$ is determined by the length of the *critical path* which is the sum of the following components:

(i) processing time of both operations of $J_{\pi(u)}$;

(ii) total processing time of all jobs that precede $J_{\pi(u)}$ on machine A;

(iii) total processing time of all jobs that follow $J_{\pi(u)}$ on machine B.

Example 1.1 *We are given a set of 4 jobs J_1, J_2, J_3 and J_4 with processing times*

$$\begin{aligned} a_1 &= 4, \quad b_1 = 3; \\ a_2 &= 2, \quad b_2 = 1; \\ a_3 &= 4, \quad b_3 = 2; \\ a_4 &= 1, \quad b_4 = 2 \end{aligned}$$

Assume that schedule S_π is defined by permutation $\pi = (J_1, J_2, J_3, J_4)$, see Figure 1.1. It is easy to see that in this schedule job J_3 is critical and the critical path is $(O_{1A}, O_{2A}, O_{3A}, O_{3B}, O_{4B})$, where O_{jA} and O_{jB} are the operations of job J_j on machines A and B, respectively. The makespan of the schedule is determined by the length of this critical path and is equal to 14.

1.4.1 Johnson's Algorithms

We consider problem $F2||C_{\max}$. This problem admits a fast polynomial-time algorithm due to Johnson [68]. Formally we can describe this algorithm as follows.

Algorithm J

INPUT: Problem $F2||C_{\max}$.

OUTPUT: An optimal schedule S_J.

1. Partition the set of jobs N into two subsets N_A and N_B as follows

$$N_A = \{i \in N | a_i < b_i\} \text{ and } N_B = \{i \in N | a_i \geq b_i\}.$$

2. Form a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ in which all jobs of set N_A precede those of set N_B; the jobs of set N_A are sequenced in non-decreasing order of a_j while the jobs of set N_B are sequenced in non-increasing order of b_j.

3. Denote the schedule associated with permutation π by S_J and stop.

Since Step 2 of Algorithm J involves the sorting of jobs according to their processing times, this step determines the overall complexity. Hence, Algorithm J requires $O(n \log n)$ time. The permutation π found by the algorithm is called a *Johnson permutation* of set N of jobs. If the sequence of jobs is a Johnson permutation then we will say that the sequence obeys the *Johnson rule.*

Johnson [68] shows that there exists an optimal schedule associated with a permutation in which job J_k precedes job J_l if the condition

$$\min\{a_k, b_l\} \leq \min\{a_l, b_k\} \tag{1.2}$$

is satisfied. It is easy to see that the condition (1.2) is transitive and Algorithm J in fact orders all jobs according to this inequality.

1.5 Flow Shop No-wait Problem

We are given a set of jobs $N = \{J_1, J_2, \ldots, J_n\}$ and m machines M_i, $i = 1, 2, \ldots, m$. Each job $J_j \in N$ consists of the set of operations $\{O_{1j}, \ldots, O_{mj}\}$, where O_{ij} denotes an operation of job J_j on machine M_i, the processing time

of each operation O_{ij} is known in advance and is denoted by p_{ij}. All jobs are assigned to the same processing route $(M_1, M_2, \ldots, M_m)$ of machines and each job once started has to be processed on all machines without interruptions and inter-stage delays until it is completed.

Goyal and Sriskandarajah [56] and Hall and Sriskandarajah [62] provide thorough surveys of complexity and algorithms for no-wait scheduling.

Two variants of the flow shop no-wait problem are considered in the literature regarding the way the operations with zero processing time are treated. The first variant assumes that if the processing time of operation O_{ij} of job J_j on machine M_i is equal to zero, i.e., $p_{ij} = 0$, then this job is not processed on machine M_i and the corresponding operation is called a *missing* operation. The other interpretation of expression $p_{ij} = 0$ implies in fact that $p_{ij} = \varepsilon > 0$, where ε is a sufficiently small value and can be disregarded. For the no-wait flow shop scheduling, the choice of interpretation of zero processing times appears to be essential.

Sahni and Cho [129] show that $F2|no{-}wait|C_{\max}$ with missing operations in the first stage is NP-hard in the strong sense. It is straightforward to observe that a symmetric problem with missing operations in the second stage is strongly NP-hard as well.

For the two-machine flow shop problem with missing operations Glass et al. [49] describe special cases which are polynomially solvable. For the case with missing operations on the second machine they derive 4/3-approximation algorithm which requires $O(n \log n)$ time.

The complexity status of the no-wait flow shop, in which zero processing times are treated as positive but negligibly small values, is different. In this case, any no-wait flow shop schedule is easily seen to belong to the class of permutation schedules.

Piehler [116] shows that problem $F|no{-}wait|C_{\max}$ can be reduced to the special case of the asymmetric travelling salesman problem. This asymmetric TSP has a special structure of the distance matrix and can by solved to

optimally by the algorithm of Gilmore and Gomory [46] in the case of two machines. This approach has been further developed by Gilmore et al. [47] and Papadimitriou and Kannelakis [114] and the original quadratic time algorithm is improved and yields an $O(n \log n)$-time algorithm. Later, Rote and Woeginger [120] show that the time complexity $O(n \log n)$ in fact is the best possible; they also develop a fully-polynomial time approximation scheme with linear running time $O(n \log \frac{1}{\varepsilon})$ for this problem.

Röck establishes NP-hardness in the strong sense of problem $F3|no-wait|C_{\max}$, see [122]. This result improves the previously known complexity result by Papadimitriou and Kanellakis [114] for the four-machine flow shop problem with the no-wait constraint. Röck [123] proves that problems $F2|no{-}wait|L_{\max}$ and $F2|no{-}wait|\sum C_j$ are NP-hard.

Sviridenko and Woeginger [140] present a PTAS for the permutation flow shop no-wait problem with a fixed number of machines.

1.6 Flow Shop No-wait Algorithms

The algorithm for the flow shop no-wait problem is due to Gilmore and Gomory. Their algorithm uses the fact that this problem can be transformed into a special kind of the Travelling Salesman Problem (TSP) which admits a polynomial-time algorithm.

1.6.1 The Travelling Salesman Problem

We can formulate formally the TSP as follows. We are given a graph with n vertices and all vertices are connected with each other by arcs. A closed route that visits each of the vertices exactly once is called a *Hamiltonian tour*. For a matrix of distances, which is not necessarily symmetric, find a Hamiltonian cycle of minimal length in this graph.

The TSP is known to be NP-hard. Since many combinatorial problems can be transformed into the TSP, special well-solvable cases of the problem

are constantly of interest. See Gilmore et al. [47] and Burkard et al. [21] for surveys of well-solvable cases of the problem. Perhaps the most well-known polynomially solvable variant of the TSP is the case identified by Gilmore and Gomory [46], see Section 1.6.3 for the detailed discussion. Kabadi and Baki [69] study the properties of this approach and have shown that it can be generalized and applied to a larger class of the TSP.

An important relaxation of the TSP is the so-called *assignment* problem. The main difference between the TSP and the assignment problem is that subtours are allowed in the solution of the latter problem. Formally it can be formulated as follows.

$$
\begin{aligned}
\text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} D_{ij}x_{ij} \rightarrow \min; \\
& \sum_{j=1}^{n} x_{ij} = 1, \ i = 1,2,\ldots,n; \\
& \sum_{i=1}^{n} x_{ij} = 1, \ j = 1,2,\ldots,n; \\
& x_{ij} \in \{0,1\}, \ i = 1,2,\ldots,n, \ i = 1,2,\ldots,n,
\end{aligned}
$$

where x_{ij} indicates whether or not vertices i and j are connected by the arc (i,j).

1.6.2 Trasformation of $F2|no{-}wait|C_{\max}$ to the TSP

The flow shop no-wait scheduling problem can be transformed into a special case of the asymmetric TSP. To do this we add a dummy job J_0 with zero processing times on all machines to the given instance of the problem. Then associating a vertex with each job we construct a graph G with the matrix $D = (D_{ij})_{(n+1)\times(n+1)}$ of distances between the vertices defined by formula

$$
D_{ij} = \max_{q=1,\ldots,m} \left\{ \sum_{k=q}^{m} p_{kj} - \sum_{k=q+1}^{m} p_{ki} \right\}.
$$

It is necessary to point out that in general $D_{ij} \neq D_{ji}$.

It is possible to show that every feasible permutation schedule of the original flow shop no-wait problem corresponds to a directed Hamiltonian cycle in this graph G and the length of the schedule is equal to the length of this cycle. Conversely, after deleting the vertex associated with the dummy job J_0 from a Hamiltonian cycle we obtain a Hamiltonian path that corresponds to a feasible schedule of the same length.

For problem $F2|no-wait|C_{\max}$ the makespan of a schedule S associated with a given permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of job indices can be rewritten as

$$C_{\max}(S) = a_{\pi(1)} + \sum_{k=1}^{n-1} \max\{a_{\pi(k+1)} - b_{\pi(k)}, 0\} + \sum_{k=1}^{n} b_k. \tag{1.3}$$

Since the last term in (1.3) is constant for any given instance of the problem define

$$\hat{C}(\pi) = a_{\pi(1)} + \sum_{k=1}^{n-1} \max\{a_{\pi(k+1)} - b_{\pi(k)}, 0\}. \tag{1.4}$$

It follows that in order to minimise $C_{\max}(S)$ it suffices to minimise $\hat{C}(\pi)$ over the set of all permutations of the job indices.

Introduce the TSP with $n+1$ vertices numbered by integers $0, 1, \ldots, n$. Recall that in the TSP it is required to find a tour of minimum length. Denote the distance between vertices p and q by D_{pq}, where $p = 0, 1, \ldots, n$; $q = 0, 1, \ldots, n$; $p \neq q$, and let $\tau = (\tau(0), \tau(1), \ldots, \tau(n))$ be a Hamiltonian tour. Then the length $D(\tau)$ of that tour can be expressed by

$$D(\tau) = \sum_{k=1}^{n} D_{\tau(k-1),\tau(k)} + D_{\tau(n),\tau(0)}. \tag{1.5}$$

We assume that city j corresponds to job $J_j, j = 1, \ldots, n$, of the original flow shop no-wait problem.

If we define

$$\begin{aligned} D_{pp} &= +\infty,\ p = 0, 1, \ldots, n; \\ D_{0q} &= a_q,\ q = 1, \ldots, n; \\ D_{pq} &= \max\{a_q - b_p, 0\},\ p = 1, \ldots, n;\ q = 1, \ldots, n;\ p \neq q, \\ D_{p0} &= 0,\ p = 1, \ldots, n. \end{aligned} \tag{1.6}$$

then (1.4) and (1.5) will coincide. This implies that permutation $\pi^* = (\pi^*(1), \pi^*(2), \ldots, \pi^*(n))$ specifies an optimal schedule for problem $F2|no - wait|C_{\max}$ if and only if the permutation $\tau^* = (0, \pi^*(1), \pi^*(2), \ldots, \pi^*(n))$ is an optimal tour for the TSP with the matrix of the form (1.6).

Further, the matrix (1.6) satisfies the Gilmore-Gomory conditions:

$$D_{pq} = \begin{cases} \int_{\beta_p}^{\alpha_q} u(x)dx, \alpha_q \geq \beta_p; \\ \\ \int_{\alpha_q}^{\beta_p} v(x)dx, \alpha_q < \beta_p, \end{cases} \tag{1.7}$$

$$p = 0, 1, \ldots, n;\ q = 0, 1, \ldots, n;\ p \neq q, \tag{1.8}$$

where u and v are integrable functions such that $u(x) + v(x) \geq 0$. To see this, we may define $\alpha_0 = \beta_0 = 0$; $\alpha_p = a_p, \beta_p = b_p$ for $p = 1, 2, \ldots, n$, and $u(x) \equiv 1$, $v(x) \equiv 0$.

Hence we have shown that the two-machine no-wait flow shop problem can be transformed to the asymmetric TSP problem which satisfies the Gilmore-Gomory conditions and consequently can be solved by the algorithm described in Section 1.6.3.

1.6.3 Gilmore-Gomory Algorithm

Gilmore and Gomory [46] consider the asymmetric TSP with the special structure of the distance matrix. Assume that distance D_{pq} between vertices i and j is defined in Section 1.6.2. The algorithm uses patching subtours

found as a solution of the associated assignment problem and finally merges them into an optimal complete tour.

Algorithm GG

INPUT: A TSP problem satisfying (1.7).

OUTPUT: An optimal permutation π^*.

1. Renumber all the vertices in such a way that $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_n$ and find permutation $\phi = (r_1, r_2, \ldots, r_n)$ such that $\alpha_{r_1} \leq \alpha_{r_2} \leq \ldots \leq \alpha_{r_n}$. Define $\gamma_i = \min\{\beta_i, a_{r_i}\}$ and $\delta_i = \max\{\beta_i, a_{r_i}\}$ for all $i = 1, 2, \ldots, n$.

2. Define function $c_\phi(i)$ such that

$$c_\phi(i) = \begin{cases} \int_{\delta_i}^{\gamma_{i+1}} (u(x) + v(x))dx, & \delta_i < \gamma_{i+1}; \\ 0, & \text{otherwise.} \end{cases}$$

3. Construct non-oriented graph G_ϕ with n vertices numbered $1, 2, \ldots, n$ such that vertex i is adjacent to vertex j if and only if $i = r_j$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$. In fact this graph is a solution of the associated assignment problem.

4. If graph G_ϕ is connected then permutation ϕ is the solution of the considered TSP. Otherwise, construct graph G'_ϕ such that each connected component of graph G_ϕ is replaced by a new vertex. If vertices i and $i+1$ belong to different components of connectivity in graph G_ϕ then vertices of graph G'_ϕ corresponding to these two components are connected with new edge R_i with length $c_\phi(i)$ for all $i = 1, 2, \ldots, n-1$.

5. Find a minimal spanning tree in graph G'_ϕ. Edges which are included into the minimal spanning tree we split into two groups. Edge R_i belongs to the first group if $\alpha_{r_i} < \beta_i$, otherwise, the edge belongs to the second group.

6. Let the number of edges in the first group be equal to $l \geq 1$. Define j_1 as the maximal value i such that edge R_i belongs to the first group. Analogously for the rest edges of the first group we find the values of $j_2, j_3, \ldots, j_l$. Let the number of edges in the second group be equal to $h \geq 1$. Define j_{l+1} as the minimal value i such that edge R_i belongs to the second group. Analogously for the rest edges of the second group we find the values of $j_{l+2}, j_{l+3}, \ldots, j_{l+h}$.

7. Consider permutation $(1, 2, \ldots, n)$. Interchange elements j_1 and $j_1 + 1$. In the obtained permutation interchange elements j_2 and $j_2 + 1$ and so on until we interchange elements j_{l+h} and $j_{l+h} + 1$.

8. In the found permutation interchange element i by element r_i for $i = 1, 2, \ldots, n$ according to permutation ϕ. Denote the obtained permutation by $\tilde{\pi} = (i_1, i_2, \ldots, i_n)$. Construct finally permutation $\pi^* = (1, j_1^*, j_2^*, \ldots, j_n^*)$ assuming that $j_1^* = i_1$, $j_k^* = i_{j_{k-1}^*}$ for all $k = 2, 3, \ldots, n$. Output permutation π^* and stop.

We notice that the algorithm in first three steps solves the assignment problem (matching) for the original matrix, and then merges the obtained partial tours into a complete optimal tour (patching). For Gilmore-Gomory matrices the assignment problem can be solved in $O(n \log n)$ time by a suitable sorting of α_k's and β_k's, while all patching steps can be implemented in linear time; see, e.g., Burkard et al. [21] and Gilmore et al. [47].

Remark 1.1 *Knowing a solution for a particular instance of the problem with a given set of jobs one can use it for obtaining a solution for any subset of jobs. Suppose that a solution to some instance of problem* $F2|no-wait|C_{\max}$ *is found. If now some subset* Q *of jobs is removed from the instance, an optimal solution does not have to be sought from scratch. It suffices to delete the jobs of* Q *from the solution of the matching subproblem with the full set*

of jobs and then to perform the patching of the remaining subtours. The new solution can therefore be found in $O(n)$ time.

1.7 Open Shop Problem

Let us recall that in the open shop problem, each job has to be processed on each machine, but the machine routes for each job are not specified in advance and have to be chosen. Formally, there are m machines M_i, $i = 1, 2, \ldots, m$, in the system and the set of jobs $N = \{J_1, J_2, \ldots, J_n\}$. Each job $J_j \in N$ can be viewed as the set of operations $\{O_{1j}, O_{2j} \ldots, O_{mj}\}$, where O_{ij} denotes an operation of job J_j on machine M_i, the processing time of each operation O_{ij} is known in advance and is denoted by p_{ij}.

An apparent lower bound on the value of the makespan for the open shop problem to minimise the makespan is

$$LB = \max \left\{ \max_{1 \le i \le m} \sum_{j \in N} p_{ij}, \max_{j \in N} \sum_{i=1}^{m} p_{ij} \right\}. \tag{1.9}$$

In other words, the optimal makespan is not less than the largest machine workload or the largest total job processing time and these bounds are called a *machine-based* lower bound and a *job-based* lower bound, respectively. Gonzalez and Sahni [52] show that for $O2||C_{\max}$ the lower bound (1.9) is attainable, and provide a linear time algorithm which solves the problem, see Section 1.8.1 for the description of their algorithm.

Since Gonzalez and Sahni [52] prove that $O3||C_{\max}$ is NP-hard in the ordinary sense, there is a little hope to find a polynomial-time algorithm for the non-preemptive open shop problem for more than three processing machines. The general problem $O||C_{\max}$ is known to be NP-hard in the strong sense, see survey by Graham et al. [57]. Moreover, as shown in [150], it is NP-complete to verify whether there exists a schedule of length 4 for an instance of $O||C_{\max}$ where all processing times are integer. So far, it remains an open question whether problem $O3||C_{\max}$ is strongly NP-hard.

Other open shop problems that are known to be NP-hard in the strong sense are $O2|r_j|C_{\max}$, $O2||L_{\max}$, see [85], and $O2||\sum C_j$, see [1]. For the $O2||L_{\max}$ problem when preemptions are allowed on one machine only Borodich [15] gives an algorithm which requires $O(n^3)$ time.

In the no-wait environment, the jobs must be processed from their start to their completion without any delays between machines. Sahni and Cho [129] show that $O2|p_{ij} \geq 0, no-wait|C_{\max}$ and $O2|p_{ij} > 0, no-wait|C_{\max}$ are NP-hard in the strong sense. Other no-wait open shop problems that are known to be strongly NP-hard are $O2|p_{ij} \geq 0, no-wait|\sum C_j$, see [1], $O2|p_{ij} > 0, no-wait|\sum C_j$, see [75].

Shakhlevich and Strusevich [135] consider a more general variant of the non-preemptive two-machine open shop problem to minimise an arbitrary monotone non-decreasing function f of two arguments: completion time on machine A and on machine B (C_A and C_B, respectively). They prove that this problem can be solved optimally in linear time. Later, van den Akker et al. [4] propose an elegant result which says that there exist at most two Pareto optimal points (C_A, C_B) for which there exists a feasible schedule meeting C_A and C_B. Recall that a solution is called a *Pareto-optimal* (or efficient) solution, if there is no other solution for which at least one criterion has a better value while values of the remaining criteria are the same or better. In other words, one cannot improve any criterion without deteriorating a value of at least one other criterion.

Sevastianov and Woeginger [134] consider problem $Om||C_{\max}$ and show that it admits a polynomial time approximation scheme (PTAS), i.e., there exists $(1+\varepsilon)$-approximation algorithm for any given $\varepsilon > 0$ and the running time of this algorithm is polynomial for any fixed m and ε. The only more possible stronger result for this problem is the existence of a fully-polynomial time approximation scheme, i.e., in our case $(1+\varepsilon)$-approximation algorithm which depends on ε polynomially, but this question still remains open.

Ráczmany observed that a greedy algorithm, see Section 1.8.2, delivers a

heuristic schedule for any instance of problem $Om||C_{\max}$ with the makespan which is at most 2 times the optimal makespan. This result was reported by Bárány and Fiala [11]. It is conjectured that the greedy algorithm has the worst-case performance ratio of $(2 - 1/m)$. Chen and Strusevich [26] prove this conjecture for $m \leq 3$ and Chen and Yu [28] prove it for $m = 4$.

Williamson et al. [150] prove that for problem $O||C_{\max}$ it is impossible to create an approximation algorithm with a worst-case performance better than 5/4, unless P=NP. Hoogeveen et al. [65] prove that problem $O||\sum C_j$ does not possess a PTAS, unless P=NP.

Since (1.9) is also a lower bound for the corresponding preemptive problem, a non-preemptive schedule that attains this bound provides an optimal solution for the preemptive problem. Hence, problem $O2|pmtn|C_{\max}$ can be solved in linear time. Lawler et al. [85] propose polynomial-time algorithms for problems $O2|pmtn|L_{\max}$ and $O2|r_j, pmtn|C_{\max}$. Du and Leung [37] prove that the sum of completion times problem $O2|pmtn|\sum C_j$ is NP-hard in the ordinary sense, and Liu and Bulfin [99] show that problem $O3|pmtn|\sum C_j$ is strongly NP-hard. Problem $O2|pmtn, \overline{d_j}|\sum C_j$ with the constraint that all job have to be completed by their respective deadlines $\overline{d_j}$ is shown to be NP-hard in the strong sense, see [99]. Problem $O2|pmtn|\sum U_j$ is binary NP-hard (see [14], [75] and [85]). Moreover, as it is shown in [14], the latter problem remains NP-hard even if the jobs have a common due date (so that $d_j = d$). Sriskandarajah and Wagneur [138] show that problem $O2|pmtn, r_j|\sum C_j$ is NP-hard in the strong sense. In [48], Gladky presents another, more simplified, proof of the latter fact.

The first algorithm for solving problem $O|pmtn|C_{\max}$ had been described before the term "open shop" was introduced. In the paper by de Werra [149] problem $O|pmtn|C_{\max}$ was actually formulated and was reduced to a problem of finding the optimal edge coloring of a bipartite multigraph. This approach has been further developed by Gabow and Kariv [43]. They introduce a more general definition of the edge coloring of a multigraph and present an

algorithm of finding such a generalized coloring of a bipartite multigraph. They show that this algorithm requires $O((n+m)r\log p_{\max})$ time, where r is the number of non-zero operations and $p_{\max}$ is the maximum operation processing time.

Gonzalez and Sahni [52] propose another approach to solving the preemptive open shop problem. This approach involves the Birkhoff-von Neumann theorem on double stochastic matrices, see [12] and [105]. This approach was developed in [51] and [84] and the best algorithm has the running time of $O(r+\min\{m^4, n^4, r^2\})$.

For problem $O|pmtn, r_j|L_{\max}$ Cho and Sahni [33] derive a polynomial time algorithm. This algorithm constructs preemptive schedule that often *mix* the operations of jobs, i.e., one operation is preempted, and before this operation is resumed and completed, another operation of the same job is started and preempted, and so on. If the mixing of operations is forbidden (*no-pass* constraint), then problem $O3|pmtn, no-pass|C_{\max}$ becomes NP-hard in the ordinary sense, see [33].

1.8 Open Shop Algorithms

1.8.1 Gonzalez and Sahni's Algorithm

We consider the two-machine open shop problem denoted by $O2||C_{\max}$. Here, all jobs of set N have to be processed on the two machines A and B. The processing routes of the jobs are not given and are part of the decision-making. This problem admits a fast linear-time algorithm due to Gonzalez and Sahni [52].

Let $\pi(N)$ be an arbitrary permutation of set N. Then we can formally describe their algorithm as follows.

Algorithm GS

INPUT: An instance of problem $O2||C_{\max}$.

OUTPUT: An optimal schedule S_{GS}.

1. Split the set of jobs N into two subsets

$$N_A = \{i \in N | a_i < b_i\} \text{ and } N_B = \{i \in N | a_i \geq b_i\}.$$

2. If $N_A \neq \emptyset$, select a job $J_l \in N_A$ such that $b_l \geq \max\{a_i | i \in N_A\}$, otherwise, set $\{J_l\} = \emptyset$.

3. If $N_B \neq \emptyset$, select a job $J_r \in N_A$ such that $b_r \geq \max\{a_i | i \in N_A\}$, otherwise, set $\{J_r\} = \emptyset$.

4. If the inequality

$$a(N \backslash \{J_l\}) \geq b(N \backslash \{J_r\}) \tag{1.10}$$

holds then the form of an optimal schedule is such that machine A processes the jobs in sequence

$$(J_r, \pi(N_B \backslash \{J_r\}), \pi(N_A \backslash \{J_l\}), J_l)$$

starting at time zero and machine B processes the jobs in sequence

$$(\pi(N_B \backslash \{J_r\}), \pi(N_A \backslash \{J_l\}), J_l, J_r)$$

starting at time $\max\{a_r + b_r - b(N), 0\}$. Both machines process the jobs without any unnecessary idle time.

5. If, otherwise, (1.10) does not hold, then the form of an optimal schedule is such that machine A processes the jobs in sequence

$$(\pi(N_A \backslash \{J_l\}), \pi(N_B \backslash \{J_r\}), J_r, J_l)$$

starting at time $\max\{a_l + b_l - a(N), 0\}$ and machine B processes the jobs in sequence

$$(J_l, \pi(N_A \backslash \{J_l\}), \pi(N_B \backslash \{J_r\}), J_r)$$

starting at time zero. Both machines process the jobs without any unnecessary idle time.

6. Call the obtained schedule S_{GS} and stop.

The optimality of the obtained schedule S_{GS} is ensured by the lower bound (1.9) on the makespan of an optimal schedule. In fact, this lower bound means that the length of an optimal schedule is not less than the maximum workload and the largest processing time of a job. Algorithm GS outputs the schedule for which this bound is attained. For the output schedule S_{GS} we have that

$$C_{\max}(S_{GS}) = \begin{cases} \max\{a(N), b(N), a_l + b_l\}, & \text{if } a(N\backslash\{J_l\}) < b(N\backslash\{J_r\}), \\ \max\{a(N), b(N), a_r + b_r\}, & \text{otherwise.} \end{cases}$$

This algorithm is widely used as a base for designing heuristics for the open shop problem, see [121] for details.

1.8.2 Greedy Algorithms

A feasible open shop schedule S is called *dense* if any machine is idle if and only if there is no job which can be processed on that machine. In a schedule found by the greedy algorithm no machine stands idle if there is a job ready to be processed on it and, therefore, the resulting schedule is dense. The first greedy algorithm for the open shop problem was introduced by Ráczmany and reported by Bárány and Fiala [11]. Formally the Greedy algorithm for finding a dense schedule for the open shop problem can be implemented as follows.

Algorithm Greedy

INPUT: An instance of problem $Om||C_{\max}$.

OUTPUT: A heuristic schedule S_H.

1. For each machine M_i, $i = 1, 2, \ldots, m$, define list L_i of operations to be processed on the machine as $(O_{i1}, O_{i2}, \ldots, O_{in})$.

2. At any time, when some machine M_i becomes available, scan the list L_i to find the first job on the list, say job J_k, that may start on M_i

earlier than the other jobs. Assign J_k to be processed on M_i starting at the earliest possible time. Remove operation O_{ik} from L_i. If the same job may simultaneously start on several machines, give preference to the machine with the smaller index i.

3. Repeat Step 2 until all lists L_i are empty.

4. Call the obtained schedule S_H and stop.

If implemented literally, the standard greedy algorithm requires looking through the whole list of allowed operations on each machine, which requires $O(n^2 \min\{n, m\})$ time, where n is the number of jobs and m is the number of machines. Aksjonov [5] presents a greedy algorithm for finding a dense open shop schedule which requires $O(nm \min\{n, m\})$. This algorithm provides a heuristic schedule with the worst-case ratio bound of 2. Clearly that this algorithm becomes linear for any fixed number of machines. The greedy approach has been intensively studied in the literature, see, e.g., [73] and [133].

Bárány and Fiala [11] report the following result due to Ráczmany.

Theorem 1.1 *For problem* $Om||C_{\max}$ *any dense schedule* S_D *guarantees that for any* m *inequality*

$$C_{\max}(S_D) - C_{\max}(S^*) \leq (m-1)\, p_{\max}$$

holds and this bound is tight.

The following result for the worst-case ratio bound of dense schedules was independently proved by Aksjonov [5] and by Wein [148].

Theorem 1.2 *For problem* $Om||C_{\max}$ *any dense schedule* S_D *guarantees that for any* m *inequality*

$$\frac{C_{\max}(S_D)}{C_{\max}(S^*)} < 2 \tag{1.11}$$

holds.

The tightness of (1.11) was not proved. It is conjectured in [26] that the bound (1.11) can be improved and the following inequality holds

$$\frac{C_{\max}(S_D)}{C_{\max}(S^*)} \leq 2 - \frac{1}{m}.$$

This bound is proved for $m \leq 3$ by Chen and Strusevich [26] and for $m = 4$ by Chen and Yu [28].

An algorithm which employs a greedy approach with prearranged operation sequencing is developed in [139]. The algorithm creates an approximate solution S_H such that either

$$\frac{C_{\max}(S_H)}{C_{\max}(S^*)} \leq 2 - \frac{1}{m+1}$$

or

$$C_{\max}(S_H) \leq C_{\max}(S^*) + (m-2)p_{\max}.$$

Chen and Strusevich [26] present a linear-time algorithm that transforms a dense schedule into a heuristic schedule with a worst-case ratio bound of 3/2 for the case of 3 machines. They prove the following property of dense schedules which is used further in our argumentation.

Lemma 1.1 *If there is some idle interval on a machine then after this idle interval the machine processes no more than $m-1$ jobs in any dense schedule.*

For the two-machine case we can get a stronger result.

Lemma 1.2 *In any dense two-machine open shop schedule there exists at most one idle time interval and after this interval only one job is processed.*

Proof. Since we consider dense schedules, two machines cannot be idle simultaneously. Without loss of generality assume that the first idle time occurs on machine A before operation O_{jA} of job J_j. It may happen if and only if operation O_{jB} is being processed at this time on machine B and no other operation requires to be processed on machine A. Hence, after the

time when job J_j is completed on machine B operation O_{jA} will be the only operation which has to be processed on machine A. This implies that starting from the time C_{jB} machine B processes only the jobs which have already been processed on machine A. Hence no idle time on machine B is possible. This proves the lemma. ■

Sevastianov and Woeginger [134] use the greedy algorithm as a part of their PTAS for the multi-machine open shop. The properties of greedy algorithms for the open shop problems were widely studied in the literature, see, e.g., [27, 132, 136, 148].

1.8.3 Pinedo and Schrage Algorithm

Pinedo and Schrage [117] present an algorithm of constructing an optimal schedule for the two-machine open shop problem based on the greedy approach with reserved operations. Their algorithm first splits the set of operations into 3 subsets: fixed, reserved and non-reserved. In fact, the operation with the longest processing time becomes reserved one, the other operation of this job is added to the set of fixed operations. After that the algorithm schedules the fixed operations and then adds all non-reserved operations in a greedy manner, see Section 1.8.2 for details. Finally, the reserved operations are added to the schedule and the complete schedule is output. Formally we can describe the algorithm as follows.

Algorithm PS

INPUT: An instance of problem $O2||C_{\max}$.

OUTPUT: An optimal schedule S_{PS}.

1. Find job J_k such that

$$\max\{a_k, b_k\} = \max_{j \in N}\{a_j, b_j\}.$$

2. If $a_k > b_k$ then rename machine A and B and correspondingly interchange notions a_j and b_j.

3. On machine A start the operation of job J_k at time 0.

4. On both machines A and B schedule the operations of all jobs of set $N \setminus \{J_k\}$ in a greedy manner.

5. On machine B determine the earliest time τ when the machine becomes idle. If there exists job J_l such that its operation is processed on B after τ, then remove this operation from the current partial schedule and define the sequence $\phi = (J_k, J_l)$. Otherwise, set $\phi = (J_l, J_k)$.

6. Start the sequence of jobs ϕ on machine B at time $\max\{a_k, \tau\}$.

7. Call the obtained schedule S_{PS} and stop.

The running time of Algorithm PS is linear. For the length of the output schedule we have that

$$C_{\max}(S_{PS}) = \begin{cases} \max\{a(N), b(N)\}, & \text{if } a_k \leq \tau; \\ a_k + \max\{a(N \setminus \{J_k\}), b_k\}, & \text{otherwise.} \end{cases}$$

It is easy to see that the lower bound (1.9) on the length of the optimal makespan for the two-machine open shop is attained.

1.9 Non-approximability

In this section we briefly recall basic facts on APX-hardness. The term "APX-hardness" is used now in the literature instead of the term "Max SNP-hardness". Papadimitriou and Yannakakis [115] study the possibility of creating a polynomial time approximation scheme (PTAS) for NP-hard problems. They show that there exists a class of APX-hard problems and such problems do not admit a PTAS.

Papadimitriou and Yannakakis [115] introduce *L-reduction* which is used as the main tool for dealing with problems from this class. An L-reduction can be defined as follows.

Let P_1 and P_2 be two optimisation problems. An L-reduction from P_1 to P_2 is a pair of functions R and S, both computable in polynomial time with the following two additional properties:

1. For any instance I of P_1 with optimum $Opt(I)$, $R(I)$ is an instance of P_2 with optimum cost $Opt(R(I))$, such that

$$Opt(R(I)) \leq \alpha Opt(I),$$

for some positive constant α.

2. For any feasible solution s of $R(I)$, $S(s)$ is a feasible solution of I such that

$$|Opt(I) - c(S(s))| \leq \beta \, |Opt(R(I)) - c(s)| \,,$$

for some positive constant β, where $c(S(s))$ and $c(s)$ represent the cost of $S(s)$ and s, respectively.

In fact, Papadimitriou and Yannakakis [115] show that L-reduction has a property of preserving an approximation. In other words, if problem P_2 admits a polynomial time approximation scheme, i.e., a family of $(1+\varepsilon)$-approximation algorithms, for any fixed $\varepsilon > 0$, and if there exists L-reduction of problem P_1 to problem P_2 with parameters α and β then problem P_1 admits $(1+\alpha\beta\varepsilon)$-approximation algorithm. Hence, there exists a PTAS for problem P_1 if there exists a PTAS for problem P_2.

According to the definition of L-reduction Papadimitriou and Yannakakis [115] define a class of optimisation problems which is closed under L-reductions. The hardest problems in this class with respect to L-reduction are called APX-complete problems. Papadimitriou and Yannakakis [115] prove that every problem in this class admits a ρ-approximation algorithm, where ρ is a positive constant. An optimisation problem which is at least as hard as any APX-complete problem with respect to L-reduction is called APX-hard.

A number of optimisation problems are proved to be APX-hard, e.g., the problem of finding a maximum cut in a graph, the problem of finding the vertex cover of the maximum size in a graph. For none of these APX-hard problems a PTAS has been constructed, moreover, due to the properties of L-reduction if there exists a PTAS for at least one APX-hard problem then all other APX-hard problems admit a PTAS.

Arora et al. [8] prove the following theorem.

Theorem 1.3 *If there exists a PTAS for at least one APX-hard problem, then P=NP.*

Theorem 1.3 gives a strong tool for proving the non-approximability of an optimisation problem. To do this it suffices to provide L-reduction from an APX-hard problem to the problem under consideration, then, unless P=NP, the considered problem does not admit a PTAS.

It has to be pointed out that this method is not the only one which is used for proving non-approximability of optimisation problems. Kubiak et al. [76] introduce a very convenient method of proving non-approximability results for problems with machines non-availability constraints. This approach is widely used and can be illustrated by an example given in Section 2.1.

Chapter 2

Machine availability constraints

2.1 Introduction

As mentioned in Chapter 1, this thesis mainly studies scheduling problems with machine availability constraints, i.e., we assume that the machines may become non-available throughout the planning time. Sometimes we will refer to a non-availability interval as a *hole*. Under this circumstances, the classical scheduling algorithms may become unacceptable and produce schedules which are far from optimal. Thereby, special consideration of such problems is required in order to obtain optimal or near-optimal feasible schedules.

We now point out some situations when non-availability intervals may occur on machines and which properties these intervals have. At the beginning of the scheduling period one or several of the machines may still continue processing some jobs scheduled in the previous time horizon. In such a situation the structure of the non-availability intervals may be quite complicated and the starting and ending times of each non-availability interval are known in advance and cannot be changed. Another possible situation arises when one or several machines require some maintenance throughout the planning period. The maintenance decision may be done separately or jointly with the job scheduling. In the first case we obtain a situation similar to the one with the overlapping of the planning periods in which we cannot control the start-

ing and ending times of these non-availability intervals. In the latter case the starting times of the non-availability intervals and possibly their lengths are not given in advance and become part of the decision-making. Another example of machine non-availability can be a consequence of machine breakdowns. It is clear that in this case we are not usually given information regarding the starting times or lengths of such non-availability intervals.

In this chapter we give a review of the present achievements in scheduling with availability constraints. Creating a classification of such problems we first split all variety of problems into three groups regarding machine environment: a single machine, parallel machines and multi-stage systems. Then for each of these groups we recall known results for different objective functions and for various patterns of non-availability intervals.

Following Lee [90], we study various scenarios of handling a non-availability interval. If some operation cannot be completed on a certain machine before a non-availability interval then we will refer to such an operation as *affected* by the non-availability interval and we will call this job a *crossover* job. Lee [90] calls the scheduling model *resumable*, if the total processing time of the operation interrupted by a non-availability interval remains equal to its original processing time, i.e., the processing of the affected operation is interrupted by the hole and is resumed when the machine becomes available again. The model is called *non-resumable*, if the total processing time of the affected operation after the hole is equal to its original processing time, as if the operation restarts from scratch. In the *semi-resumable* model, the fragment of an operation performed before the hole has to be partially reprocessed after the hole. Thus, under the semi-resumable scenario, the total processing time of an operation becomes greater than its original processing time.

Assuming that a hole occupies interval $[s, t]$, operation O_j is affected and its processing time is equal to p_j we formally describe the above scenarios as follows.

Resumable Scenario. In this case the processing of operation O_j is interrupted at time s and resumed at time t, the total processing time remains equal to p_j. This scenario can be applied to a typist who has gone to a lunch break and then resumes the typing of a manuscript from the last typed page.

Semi-Resumable Scenario. Let x_j be the duration of the processing of operation $O_{j,L}$ before time s. Under the semi-resumable scenario, we are given a job-dependent value of $\alpha_j \in [0, 1]$, such that a part of the operation performed before the hole must be done again for $\alpha_j x_j$ time units to reach the status at the point of interruption. Then the processing is resumed for $p_j - x_j$ time units, so that the total processing time of the operation after the hole is equal to $p_j - (1 - \alpha_j)x_j$. Notice that if all $\alpha_j = 0$ the resumable and the semi-resumable scenarios are equivalent. Our definition of the semi-resumable scenario is slightly more general than that introduced by Lee [90] and used so far in the literature, where the values $\alpha_j = \alpha$ for all jobs. This scenario is applicable if, e.g., the processing machine is a heating one, so that the part being heated will cool down during the non-availability period and will have to be re-heated to the temperature at the point of interruption, but not from the original temperature.

Non-Resumable Scenario. Under this scenario, the total processing time of operation O_j after the hole is equal to p_j, as if the operation restarts from scratch. Thus, this scenario is a special case of the semi-resumable scenario when an operation has to be completely reprocessed (i.e., $\alpha_j = 1$ for all $j = 1, \ldots, n$). This scenario can be found in the situations related to downloading files from the Internet: if the connection is lost during the download, one has to start the process again when the connection is restored.

We denote the scheduling problems with non-availability constraints extending the standard three-field notation introduced by Graham et al. [57]. We add two extra parameters into the second field $\circ|\circ, H, Sc|\circ$, where $H = h(q_1, \ldots, q_m)$ refer to the non-availability pattern, where q_i holes occur on machine $i \in M$; $Sc \in \{Re, S\text{-}Re, N\text{-}Re\}$ to refer to the resumable,

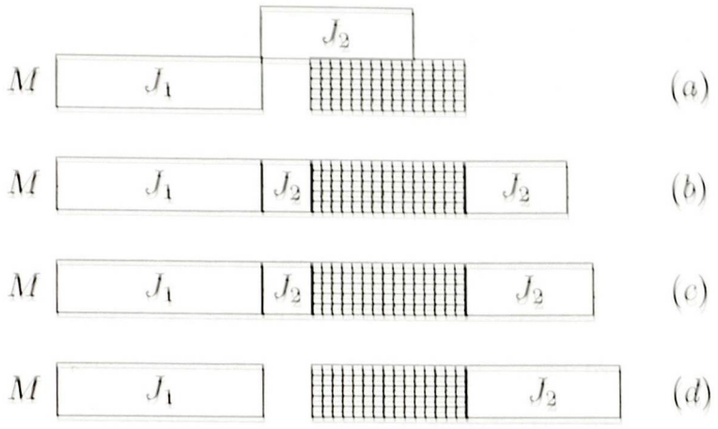

Figure 2.1: (a) Job J_2 does not fit in the gap before the hole; (b) resumable scenario; (c) semi-resumable scenario; (d) non-resumable scenario

semi-resumable or non-resumable scenario, respectively.

The following example illustrates all this scenarios of the behaviour of the affected operation.

Example 2.1 *We are given one machine M and two jobs J_1 and J_2 with processing times 4 and 3, respectively. The non-availability interval occupies the interval $[5, 8]$ and for the semi-resumable scenario constants α_1 and α_2 are equal to 0.5. Consider schedule S specified by permutation of jobs (J_1, J_2). Figure 2.1 demonstrates all scenarios.*

The most common technique of proving non-approximability of a scheduling problem with non-availability constraints is due to Kubiak et al. [76]. This approach is demonstrated in the following example.

Breit et al. [19] consider a single machine scheduling problem to minimise the makespan under the non-resumable scenario.

Theorem 2.1 *(by Breit et al. [19]) For a fixed $\rho > 1$, the existence of a polynomial-time approximation algorithm for problem $1\,|h(2), N{-}Re|\,C_{\max}$ implies that P=NP.*

Proof. We show that this algorithm, if existed, would solve optimally an NP-hard problem $1\,|h(1), N{-}Re|\,C_{\max}$, see [88]. Consider an arbitrary instance of the decision version of problem $1\,|h(1), N{-}Re|\,C_{\max}$ in which it is required to verify whether there exists a schedule S such that $C_{\max}(S) \leq y$ for a given y. Let S_1^* be a schedule that is optimal for this problem with a single hole.

Define an instance of problem $1\,|h(2), N{-}Re|\,C_{\max}$ which is obtained from the taken instance of problem $1\,|h(1)|\,C_{\max}$ by inserting an extra hole $[y, \rho y]$. Let S_2^* be an optimal schedule for this latter problem with two non-availability intervals.

If $C_{\max}(S_1^*) \leq y$ then $C_{\max}(S_1^*) = C_{\max}(S_2^*)$; otherwise, the jobs that are processed after time y in schedule S_1^* have to be processed after time ρy in schedule S_2^*. Since no preemption is allowed, it follows that $C_{\max}(S_2^*) \geq \rho y + C_{\max}(S_1^*) - y$.

Apply our algorithm to the defined instance of problem $1\,|h(2), N{-}Re|\,C_{\max}$. It will find schedule S_H such that

$$\frac{C_{\max}(S_H)}{C_{\max}(S_2^*)} \leq \rho.$$

We show that by verifying the value of $C_{\max}(S_H)$ it is possible to solve the decision version of problem $1\,|h(1), N{-}Re|\,C_{\max}$.

Suppose that $C_{\max}(S_H) \leq \rho y$. Then the actual completion time of the schedule is before the second hole, i.e., $C_{\max}(S_H) \leq y$. It is obvious then that

$$C_{\max}(S_1^*) \leq C_{\max}(S_2^*) \leq C_{\max}(S_H) \leq y,$$

so that in problem $1\,|h(1), N{-}Re|\,C_{\max}$ the required schedule exists.

Suppose now that $C_{\max}(S_H) > \rho y$. Then the inequality $C_{\max}(S_2^*) \leq y$ would imply that

$$\frac{C_{\max}(S_H)}{C_{\max}(S_2^*)} > \frac{\rho y}{y} = \rho,$$

a contradiction. Therefore, $C_{\max}(S_1^*) > y$. If $C_{\max}(S_1^*) \leq y$, then we would have $C_{\max}(S_2^*) = C_{\max}(S_1^*) \leq y$, a contradiction. Thus, in problem $1\,|h(1), N{-}Re|\,C_{\max}$ the required schedule does not exist. ■

2.2 Single Machine

The resumable scenario for a single machine and one non-availability interval is well studied. If the objective function is that of minimising the makespan then it is easy to show that any schedule is optimal if it has no idle time, moreover a preemption is necessary only if some job cannot be totally processed before the non-availability interval, i.e. there exists a crossover job, see [88]. Lee [88] shows that the SPT rule, which was defined in Section 1.2.4, solves problem $1\,|h(1), Re|\sum C_j$ optimally, but the more general problem $1\,|h(1), Re|\sum w_j C_j$ becomes NP-hard. He presents a dynamic programming algorithm for the latter problem. Also, Lee [88] shows that an optimal solution of problem $1\,|h(1), Re|\,L_{\max}$ can be found by the EDD rule, which was defined in Section 1.2.4, and the number of late jobs can be minimised by the modified algorithm of Moore and Hodgson [103]. Notice that minimising theweighted number of the late jobs is proved to be NP-hard already for the continuously available machine, see [70].

For the non-resumable scenario the following results are obtained so far. Problem $1|h(1), N-Re|C_{\max}$ is proved to be NP-hard by Lee [88]. Moreover, he proves that $1\,|h(k), S-Re|\,C_{\max}$ and $1\,|h(k), N-Re|\,C_{\max}$ are NP-hard in the strong sense if the number of holes is part of the input. He also shows that the LPT rule, which was defined in Section 1.2.4, leads to a tight bound of 4/3. Lee [88] proves that since minimising the makespan is an NP-hard problem, minimising the maximum lateness and the number of late jobs are NP-hard as well. He proves that the algorithm of Moore and Hodgson [103] for the problem to minimise the number of late jobs leads to a worst-case error bound of $P \leq P^* + 1$, where P is the number of tardy jobs obtained by the algorithm and P^* denotes the optimal number of tardy jobs. Moreover, Lee [88] shows that the EDD rule for the problem $1\,|h(1), N-Re|\,L_{\max}$ provides the following upper bound $L_{\max} \leq L^*_{\max} + \max\limits_{j=1,\ldots,n}\{p_j\}$, where $L_{\max}$ denotes the maximum lateness obtained by the algorithm, $L^*_{\max}$ denotes the optimal

	$C_{\max}$	$\sum C_j$	$\sum \omega_j C_j$	$\sum U_j$	$\sum \omega_j U_j$	$L_{\max}$
res	polynomial	polynomial	NP	polynomial	NP	polynomial
s-res	NP	NP	NP		NP	
n-res	$5/4, NP$	$20/17, NP$				

Table 2.1: Results obtained for the single machine and one non-availability interval

maximum lateness, n is the number of jobs and p_j denotes the processing time of job J_j. Lee and Liman [93] show that the problem $1\,|h(1), N\!-\!Re|\sum C_j$ is NP-hard. Lee and Liman [93] have studied the SPT rule for this problem and have proved that this heuristic leads to a tight bound of $9/7$. Sadfi et al. [127] improve this result and propose a heuristic for the problem with a worst-case error bound of $20/17$. A pseudopolynomial-time dynamic programming algorithm is developed for the problem by Sadfi et al. [128].

Breit et al. [18] show that problems $1|h(2), N\!-\!Re|C_{\max}$ and $1|h(2), S\!-\!Re|C_{\max}$ are not approximable within a fixed factor, unless P=NP.

Since the semi-resumable scenario is a generalization of the non-resumable scenario, it is clear that all problems which are proved to be NP-hard for the non-resumable scenario remain to be NP-hard for the semi-resumable one.

2.3 Parallel Machines

The classical scheduling problem of minimising the makespan on m parallel machines without non-availability constraints is NP-hard. Lee [87] considers this problem under the assumption that each machine may not be available for processing at time zero. He proves that the LPT algorithm delivers a heuristic solution with the worst-case ratio bound of $\frac{3}{2} - \frac{1}{2m}$. Moreover he presents a modification of this algorithms with a better performance of $\frac{4}{3}$. Kellerer [72] presents a heuristic for m parallel machines and the non-resumable scenario with the worst case ratio of $\frac{5}{4}$. He [63] proves that if a LPT schedule has a latest finishing job that runs on a machine with at least

$k-1$ other jobs then the ratio of the LPT makespan to the optimal one is at most $\frac{k+1}{k} - \frac{1}{km}$ for $k > 1$ or $\frac{3}{2} - \frac{1}{2m-2}$ for $k = 1$ and these bounds are tight. Chang and Hwang [23] employ a bin-packing heuristic algorithm known as the MULTIFIT to this problem and prove that the makespan of the output schedule is bounded by $\frac{9}{7} + 2^{-l}$ times the optimal makespan, where l is the selected number of iterations of the MULTIFIT.

Kaspi and Montreuil [71] study the problem of minimising the total completion time for the system of m parallel identical machines in which machines may be non-available for processing in the beginning of the planning period. They prove that the SPT rule produces an optimal solution of the problem.

If the non-availability interval starts at a prespecified time $s > 0$ then parallel scheduling problem with m machines subject to minimise the makespan obviously remains NP-hard for each of the scenarios: resumable, semi-resumable and non-resumable. Lee [88] presents two heuristics with a worst case ratio of $\frac{3}{2} - \frac{1}{2m}$ for the resumable scenario and $\frac{m+1}{2}$ for the non-resumable scenario and both these bounds are tight.

Problems $P2\,|h(1,0), Re|\sum w_jC_j$ and $P2\,|h(1,0), N-Re|\sum w_jC_j$ are NP-hard, since the corresponding one-machine problems are proved to be NP-hard. Lee [88] presents a dynamic programming algorithm for these problems.

If preemptions are allowed and the number of available machines is a function of time then the problems of minimising the makespan and the maximal lateness are proved to be polynomially solvable, see Leung and Pinedo [98].

Other models of machine non-availability constraints were considered in the literature. The case of two identical parallel machines where one of the machines is available only for a specified period of time subject to minimise the total completion time was considered by Lee and Liman [94]. A more general case of this problem was studied by Mosheiov [104]. He assumes that the processing system consists of m machines and these machines become

available at different times. Hwang and Chang [66] consider the parallel machine scheduling problem with planned machine shutdowns and analyze the performance of the LPT algorithm for this problem. They prove that the output schedule will have the makespan as large as twice the optimal makespan and this bound is tight provided that no more than half of the machines are allowed to be shutdown simultaneously.

2.4 Multi-stage systems

Since in this thesis we study only shop sheduling problems with the objective function of minimising the makespan, we make a more detailed review of the available literature in this area.

2.4.1 Flow Shop

Without non-availability intervals the two-machine flow shop scheduling problem can be solved in polynomial time by Johnson's algorithm in $O(n \log n)$ time, see [68] and Section 1.4.1. The complexity status of the problem changes if the machines are not continuously available. Since, due to Lee [88], the $1\,|h(k), S\!-\!Re|\,C_{\max}$ and $1\,|h(k), N\!-\!Re|\,C_{\max}$ are NP-hard in the strong sense, it is easy to see that the corresponding flow shop problems are strongly NP-hard as well. Lee [89] considers problems $F2\,|h(0,1), Re|\,C_{\max}$ and $F2\,|h(1,0), Re|\,C_{\max}$ and proves that they are NP-hard. He presents a pseudopolynomial dynamic programming algorithm for these problems. Kubiak et al. [76] study the resumable variant of the problem under the assumption that the number of non-availability intervals is part of the input. They prove that the problem becomes NP-hard in the strong sense even if all non-availability intervals occur only on one of the machines. Lee [90] provides a pseudopolynomial dynamic programming algorithm to solve the problem with one hole under the semi-resumable scenario.

The complexity status of the problem has stimulated research on its ap-

proximability. It appears that the problem allows us to establish a sharp borderline between the conditions under which it is possible to design fast approximation algorithms and the conditions under which to find an approximate solution that is guranteed to be close to the optimum is not easier than to determine the optimum exactly.

Lee [90] and Kubiak et al. [76] consider the resumable two-machine flow shop problem and show that it becomes not approximable in polynomial time within a fixed factor, unless P=NP, provided that there are either two holes on the second machine or one hole on each machine.

It is known that the classical flow shop problem $Fm||C_{\max}$ is symmetric in the sense that the optimal makespan of the flow shop schedule with renamed machines equals to the makespan of an optimal schedule for the original problem, but the non-availability intervals break this symmetry. For problem $F2\,|h(1,0), Re|\,C_{\max}$ Lee [89] develops a heuristic with the worst-case ratio of $\frac{3}{2}$ and a heuristic with the worst-case ratio of $\frac{4}{3}$ for problem $F2\,|h(0,1), Re|\,C_{\max}$. For the latter problem, a $\frac{4}{3}-$approximation algorithm is due to Cheng and Wang [32]. Breit [20] proposes an improved $\frac{5}{4}-$algorithm for this problem. Recently, Ng and Kovalyov [107] propose an FPTAS for problems $F2\,|h(1,0), Re|\,C_{\max}$ and $F2\,|h(0,1), Re|\,C_{\max}$. Blazewicz et al. [13] study the problem with several non-availability intervals and develop constructive and local search heuristic algorithms for it.

For the semi-resumable scenario (and for the non-resumable scenario as well), Lee [90] gives a $\frac{3}{2}$-approximation algorithm for problem $F2\,|h(0,1), S-Re|\,C_{\max}$ and a 2-approximation algorithm for problem $F2\,|h(1,0), S-Re|\,C_{\max}$. If there are two holes, one on the first machine and the other on the second machine, and these holes are consecutive, i.e., the second hole starts exactly when the first hole ends, Cheng and Wang [31] give a $\frac{5}{3}-$approximation algorithm.

Kubiak et al. [76] show that if several non-availability intervals under the resumable scenario occur on the first machine then Johnson's rule [68]

holes	$h(0,1)$	$h(1,0)$	$h(k,0)$	$h(0,k)$	$h(1,1)$
Re	$FPTAS$ $Dyn.Progr.$	$FPTAS$ $Dyn.Progr.$	$2-app$	$non\text{-}app$	$non\text{-}app$
$S-Re$	$4/3-app$ $Dyn.Progr.$	$4/3-app$ $Dyn.Progr.$	$non\text{-}app$	$non\text{-}app$	$non\text{-}app$
$N-Re$	$4/3-app$ $Dyn.Progr.$	$4/3-app$ $Dyn.Progr.$	$non\text{-}app$	$non\text{-}app$	$non\text{-}app$

Table 2.2: Known results for the two-machine flow shop problem with availability constraints

delivers a heuristic schedule with the worst-case error bound of 2.

2.4.2 Flow Shop No-wait

Another flow shop problem considered in the literature is the flow shop problem with no-wait in process. Notice, that this problem without non-availability intervals is polynomially solvable by Gilmore-Gomory's algorithm, see [46] and Section 1.6. Only the non-resumable scenario has been considered for the problem prior to our research. First, Espinouse et al. [38, 39] show that problems $F2\,|h(1,0), N-Re|\,C_{\max}$ and $F2\,|h(0,1), N-Re|\,C_{\max}$ are NP-hard and become strongly NP-hard in the case that the number of holes is part of input.

There are two possible interpretations of this scenario if the hole occurs on machine B. One of these interpretations (we will call it Scenario B1) is essentially equivalent to the semi-resumable scenario with $\alpha_j = 1$, where $j = 1, \ldots, n$, so that operation $O_{j,B}$ starts before the hole immediately after $O_{j,A}$ is completed, is interrupted at time s and resumed from scratch at t. The other interpretation (we will refer to it as Scenario B2) is due to Espinouse [38, 39]. It assumes that operation $O_{j,B}$ does not start before the hole at all and starts either at time t or at the completion time of operation $O_{j,A}$, whichever is the latest. See Figure 2.2 for the two interpretations of the non-resumable scenario and their influence on the makespan.

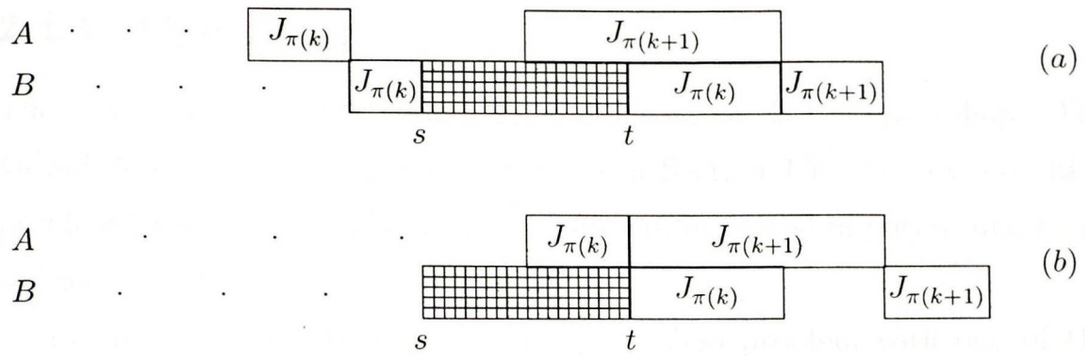

Figure 2.2: Non-resumable scenarios B1 (a) and B2 (b)

holes	$h(0,1)$	$h(1,0)$	$h(k,0)$	$h(0,k)$	$h(1,1)$
$N-Re$	$5/3-app$ $PTAS$	$5/3-app$ $PTAS$	*non-app*	*non-app*	*non-app*

Table 2.3: Known results for the two-machine flow shop no-wait problem with availability constraints

Further, along with $h(0,1)$ we may write either $Sc = N\text{-}ReB1$ or $Sc = N\text{-}ReB2$ to distinguish between the non-resumable scenarios B1 and B2.

Espinose et al. [38, 39] prove that the problem with two or more non-availability intervals is not approximable in polynomial time within any constant factor, unless P=NP. Also, they provide two heuristics for holes on the first machine and on the second one with tight error bounds of 2. This result was improved by Wang and Cheng [147]. They present two heuristics with the worst case error bounds of 5/3 for both these cases. Recently, Cheng and Liu [30] have presented a polynomial time approximation scheme for the two-machine flow shop no-wait problem for the case of a single hole either on the first or on the second machine and for the problem with one hole on each machine if these two holes overlap.

2.4.3 Open Shop

The next important multi-stage scheduling system is the open shop. The formal description of this model is given in Section 1.7. The two-machine problem without non-availability intervals can be solved in polynomial time, see Section 1.8.

Lu and Posner [101] consider the open shop problem with one of the machines unavailable at time zero and present a linear time algorithm which solves the problem to optimality.

Since we start considering the open shop problem with availability constraints, we should sort out an important difference between the resumable model studied by Breit et al. [18] and the preemptive model studied by Vairaktarakis and Sahni [143]. The resumable model is similar to the open shop with *no-pass* constraints analyzed by Cho and Sahni [33], where between the interruption of processing a job and its resumption on this machine the job cannot be processed on any other machine. In the preemptive model the interrupted job may be processed on another machine. Vairaktarakis and Sahni [143] present a polynomial time algorithm for finding an optimal solution for the preemptive open shop problem with an arbitrary number of machines and non-availability intervals. However, Breit et al. [18] and Lorigeon et al. [100] show that the two-machine resumable open shop model with one non-availability interval is NP-hard. For the non-resumable scenario since the one-machine problem with one hole is proved to be NP-hard (see [88] and Section 2.1), it follows that the two-machine open shop problem is NP-hard as well.

Breit et al. [18] show that problem $O2\,|h(k,0), Re|\,C_{\max}$ can be solved in linear time with the worst case ratio of 2. However, problem $O2\,|h(2,1), Re|\,C_{\max}$ cannot be approximated within any finite factor in polynomial time, unless P=NP. For the non-resumable scenario Breit et al. [19] prove that the two-machine open shop problem with at least two holes on

holes	$h(0,1)$	$h(0,k)$	$h(1,1)$	$h(1,2)$
Re	$4/3 - app$ $Dyn.Progr.$	$4/3 - app$		$non - app$
$N - Re$	$4/3 - app$	$non - app$	$2 - app$	$non - app$

Table 2.4: Known results for the two-machine open shop problem with availability constraints

one of the machines is non-approximable, unless P=NP.

Approximation algorithms with a worst case ratio of 4/3 was developed for problem $O2\,|h(1,0), Re|\,C_{\max}$ by Breit et al. [18]. Their algorithm is based on the Gonzalez and Sahni algorithm, see Section 1.8.1 for its description. Without modifications the algorithm of Gonzalez and Sahni may produce schedules which are arbitrarily bad, i.e. it does not produce schedules with a finite error bound ratio bound on their makespan. Depending on the starting time of the hole and the processing times of the jobs the schedule found by the Gonzalez and Sahni algorithm is appropriately modified and a heuristic schedule with the worst-case error bound of 4/3 is provided. For all cases the running time of the algorithm is linear.

The model under the non-resumable scenario was considered by Breit et al. [19]. For problem $O2\,|h(1,1), N-Re|\,C_{\max}$ they provide a linear time 2-approximation algorithm. For problem $O2\,|h(1,0), N-Re|\,C_{\max}$ they present a 4/3-approximation algorithm. This algorithm combines the greedy approach with specific scheduling of some operations.

2.5 Machine Maintenance

Another possible interpretation of machine non-availability is maintenance restrictions. Here the decision-maker can control the starting time of non-availability intervals and their lengths.

One of extensions of standard scheduling models is related to the so-called machine non-availability intervals, which have been discussed so far, and has

received considerable attention since the beginning of the 90s, see survey papers by Lee [88, 91] and Schmidt [131]. We refer to a purely deterministic interval of non-availability when the start of the interval and its duration are known in advance as *fixed*.

In equipment breakdown modeling, the decision-maker is not aware when a breakdown occurs and for how long, so that the on-line decisions have to be taken on a possible change of plans and rescheduling. This direction of research is closely related to the area of disruption management and, despite its importance, so far has not been systematically studied.

Planning the intervals of preventive equipment maintenance gives the decision-maker freedom to choose the start time for that maintenance; additionally, the length of the maintenance period may depend on its start time (the sooner the maintenance is started, the better the equipment conditions are and less time is needed for its maintenance). This calls for study of scheduling models with *floating* non-availability intervals of controllable durations. In this thesis we address scheduling problems with both types of non-availability, i.e. fixed and floating.

The importance of preventive maintenance for production enterprises and service organizations has been widely recognized by both practitioners and management scientists; see, e.g., paper by Gopalakrishnan et al. [54], the Internet emporium at www.plant-maintenance.com and popular books by Nyman and Levitt [112] and Palmer [113].

In scheduling literature there is a stream of papers dealing with finding a periodic schedule for fixed length maintenance periods. Each machine incurs an operation cost which depends on the time of the last maintenance of this machine. The problem of scheduling maintenance intervals to reduce the average long-term cost of running the system is considered by Anily et al. [6, 7]. An extended version of the problem with additional fixed maintenance cost involved is considered by Bar-Noy et al. [10] and Grigoriev et al. [59].

In another model there is an upper bound on how long a processing

machine may work without maintenance. Qi et al. [118] consider the problem of scheduling jobs and maintenance intervals of equal duration on a single machine to minimise various objective functions.

Graves and Lee [58] consider the problem of scheduling jobs and two maintenance intervals of a fixed duration on a single machine to minimise either the total weighted completion time or the maximum lateness. If the processing of a job is interrupted because of the machine maintenance, the job requires some additional setup time to resume. Lee and Chen [92] study the problem in which each parallel machine has to be maintained once during the planning period. The jobs are not allowed to be interrupted by machine maintenance. The goal is to minimise the total weighted completion times of the jobs. There are two different variations of the problem: (i) only one machine can be maintained at any time, and (ii) several machines can be maintained simultaneously.

Lee and Lin [95] consider the problem of scheduling machine maintenance in the single-machine environment. They assume that after running for a certain amount of time the machine may be in a subnormal condition in which its processing speed is reduced. The decision-maker may either stop the machine and start the maintenance work which brings the machine back in the normal condition or he or she can wait and maintain the machine later. In the case when the choice is made to continue processing the jobs the machine can break out completely, hence the repair will be required immediately. Authors assume that all jobs processing times are deterministic while the machine breakdown is a random process following certain distribution. Lee and Lin [95] consider resumable and non-resumable scenarios and various objective functions such as the expected makespan, the total expected completion time, the maximum expected lateness and expected maximum lateness.

Chapter 3

Flow Shop Scheduling

3.1 Introduction

In this chapter, we concentrate on the flow shop scheduling model which is one of the classical models for multi-stage processing systems. For detailed discussion of the flow shop problem see Section 1.3. We study the two-machine flow shop scheduling problem with various rules of treating the jobs affected by a non-availability interval and under various assumptions on the structure of the non-availability intervals.

A general discussion of the machine non-availability and a review of literature are given in Chapter 2. The material of this chapter has been reported at MAPSP'03, see [77]. An FPTAS for the two-machine flow shop problem with a single hole either on the first machine or on the second machine was independently proposed by Ng and Kovalyov [107]. Our FPTAS for a single hole on machine B is simpler than the one by Ng and Kovalev, but has the same computational complexity. Since it relies on the fact that the hole occures on machine B and it can not be easily extended to the case with a hole on machine A, we obtain an FPTAS which is worse in terms of computational complexity than one by Ng and Kovalev.

The remainder of this chapter is organized as follows. In Section 3.2, a dynamic programming algorithm for the resumable scenario and several

holes on one of the machines is presented. Section 3.3 demonstrates how to convert the available dynamic programming algorithms to FPTAS's. Since the running time of these FPTAS's is fairly large, in Section 3.4 a fast heuristic algorithm with a guaranteed worst-case ratio of 3/2 is presented for the problem with holes on the first machine. Section 3.5 describes a PTAS for the two-machine flow shop problem with a single hole on one of the machines under the semi-resumable scenario. Application of the developed method of constructing a dynamic programming algorithm to a variant of the two-machine flow shop problem is presented in Section 3.6. Some concluding remarks are given in Section 3.7.

3.2 Resumable Scenario: Dynamic Programming

In this section, we consider the two-machine flow shop scheduling problem to minimise the makespan provided that the resumable scenario is applied. We demonstrate that problem $F2|h(q,0), Re|C_{\max}$ with several non-availability intervals on the first machine is solvable by a pseudopolynomial dynamic programming algorithm.

Further, we briefly describe how to extend this approach to the problem with any fixed number of holes on each machine. This is in contrast with the strong NP-hardness of the problem with a variable number of holes, see Kubiak et al. [76], and therefore completely settles the complexity status of the resumable version of the problem under consideration.

From further consideration we exclude the situation that all jobs complete before the first hole since in this case an optimal schedule can be found by Johnson's algorithm.

Similarly to the classical flow shop problem that was considered in Section 1.3, for the two-machine flow shop problem with non-availability constraints we consider a *critical job* and a *critical path*. A critical job starts its processing

on machine B as soon as it completes its processing on machine A without any delay and the starting times of its operations on both machines cannot be delayed without increasing the total makespan. The makespan of a schedule that contains a critical job J_c is determined by the length of the critical path which is the sum of the following components:

(i) processing time of both operations of J_c;

(ii) total processing time of all jobs that precede J_c on machine A;

(iii) total processing time of all jobs that follow J_c on machine B;

(iv) the sum of lengths of all non-availability intervals on machine A before the completion time of job J_c on machine A and the total length of all non-availability intervals on machine B from this time to the time when the last job in the processing sequence completes.

It is necessary to point out that this definition of a critical path differs from the one presented in Section 1.4.1. We have to add the last term of the sum defining a critical path due to the presence of non-availability intervals which are absent in the classical model.

In the case of a single hole, a pseudopolynomial dynamic programming algorithm for problem $F2|h(1,0), Re|C_{\max}$ is designed by Lee [89]. His scheme is based on the observation that there exists an optimal schedule in which all jobs that are completed on A before the hole are sequenced according to Johnson's rule, and the rest of the jobs starting with the crossover job also form a Johnson sequence.

For problem $F2|h(q,0), Re|C_{\max}$, where $q > 1$, following the argument by Lee [89], we can restrict the search for an optimal schedule to the class of schedules in which the jobs that complete before the first hole follow a Johnson sequence. Take a feasible schedule and consider an interval I either between two consecutive holes or after the last hole such that at least two

jobs are completed in I. Notice that the first of these jobs is the crossover job J. Let t' be the left end-point of I, and $[s', t']$ be the hole that immediately precedes I. The duration of job J on A may appear to be such that J starts not in the interval that immediately precedes the hole $[s', t']$, but earlier, so that the job spans across several intervals of availability of machine A and is interrupted by several holes. Still, it is easy to verify that the argument by Lee [89] carries over, so that the following statement holds.

Lemma 3.1 *(by Lee [89]) For problem* $F2|h(q,0), Re|C_{\max}$, *where* $q \geq 1$, *there exists an optimal schedule such that for each interval of availability of machine A the jobs that complete on A in the interval follow a Johnson sequence.*

For problem $F2|h(1,0), Re|C_{\max}$, the dynamic programming algorithm by Lee [89] scans the jobs according to a Johnson sequence, assigns the next job either as the last job to be completed on A before the hole or as the last job of the current sequence. As state variables, the algorithm uses the completion times of the last job before the hole on both machines and the starting times of the crossover job on both machines.

For problem $F2|h(q,0), Re|C_{\max}$ with $q > 1$, we are unable to extend Lee's algorithm in a straightforward way, since here we have to handle the crossover jobs that may span across several consecutive availability intervals. Therefore, in the initialization stage of our algorithm, we generate all possible placements of the crossover jobs and store all relevant starting times of these jobs on both machines. In the next stage, we scan the remaining jobs according to a Johnson sequence and assign them for processing into the existing gaps of the current partial schedule.

Consider an arbitrary assignment of the crossover jobs after the initialization stage. The jobs are selected in any order and are placed around the holes on machine A, so that no job is fully processed on A in an interval between two consecutive holes (by convention, a job that starts at the right

end-point of a hole t_k is treated as if it started at the left end-point s_k of that hole and is interrupted by the hole). The job assignment is finished when no other crossover job can be placed.

Suppose that $r \leq q$ jobs have been selected as crossover jobs. Without loss of generality, renumber these jobs by $J_1, J_2, \ldots, J_r$ in the order of their appearance in the current partial schedule. For job J_k, let s^k denote the left end-point of the first hole that interrupts its processing on A and t^k denote the right end-point of the last hole after which J_k is completed on A. Denote the total length of the holes in the interval $[s^k, t^k]$ by δ^k.

For each crossover job J_k define possible starting times v_A^k and v_B^k of that job on machines A and B, respectively. Given a choice of crossover jobs and their arrangements with respect to the holes on machine A, at the initialization stage we need to enumerate all relevant integer values of v_A^k and v_B^k. By definition, v_A^k cannot be greater than s^k. Besides, to guarantee that job J_k cannot be completed on A earlier than time s^k and produces no conflicts with an earlier scheduled crossover job, we derive that $v_A^1 \geq \max\{s^1 - a_{\max} + 1, 0\}$ and $v_A^k \geq \max\{s^k - a_{\max} + 1, v_A^{k-1} + a_{k-1} + \delta^{k-1}\}$ for all $k = 2, \ldots, r$. On the other hand, $v_B^1 \geq v_A^1 + a_1 + \delta^1$ and $v_B^k \geq \max\{v_A^k + a_k + \delta^k, v_B^{k-1} + b_{k-1}\}$ for all $k = 2, \ldots, r$. In the class of schedules with the current choice and placement of the crossover jobs, the completion time of job J_k on machine B is bounded from above by $A^k + \delta^k + B^k$, where A^k denotes the total processing time of all jobs except the jobs $J_{k+1}, \ldots, J_r$ on machine A and B^k is defined analogously with respect to machine B. This implies that the number of possible integer values for v_B^k does not exceed $A^k + B^k$ for each k, or $a(N) + b(N)$ for all k.

Given one of these initial schedules defined by the choice of the crossover jobs $J_1, \ldots, J_r$ and their starting times v_A^k and v_B^k, renumber the remaining jobs so that $J_{r+1}, \ldots, J_n$ form a Johnson sequence. Consider a partial schedule that is obtained from the chosen initial schedule by assigning i jobs $J_{r+1}, \ldots, J_{r+i+1}$. Associate with this partial schedule a state of the following

form:

$$(i; v_A^1, v_B^1, \ldots, v_A^r, v_B^r; u_A^1, u_B^1, \ldots, u_A^r, u_B^r; x),$$

where

u_A^k is the completion time of the last job on machine A that finishes no later than v_A^k, $k = 1, 2, \ldots, r$;

u_B^k is the earliest possible completion time of that job on machine B, $k = 1, 2, \ldots, r$;

x is the makespan of the current partial schedule.

The algorithm scans the remaining jobs and tries to insert the next job J_{r+i+2} to be processed as the last job between two crossover jobs J_k and J_{k+1}, for $k = 1, 2, \ldots, r$. If that cannot be done, i.e., if either $u_A^k + a_{r+i+2} > v_A^{k+1}$ or $\max\left\{u_A^k + a_{r+i+2}, u_B^k\right\} + b_{r+i+2} > v_B^{k+1}$, then the resulting partial schedule becomes infeasible and is disregarded. Otherwise, the new partial schedule is associated with the state:

$$\begin{aligned}
&(i+1; v_A^1, v_B^1, , \ldots, v_A^r, v_B^r;\\
&u_A^1, u_B^1, \ldots, u_A^k + a_{r+i+2}, \max\left\{u_A^k + a_{r+i+2}, u_B^k\right\} + b_{r+i+2},\\
&u_A^{k+1}, u_B^{k+1}, \ldots, u_A^r, u_B^r; x).
\end{aligned}$$

Besides, the algorithm tries to put job J_{r+i+2} as the last job of the current sequence. In this case, the resulting partial schedule is associated with following state:

$$\left(i+1; v_A^1, v_B^1, \ldots, v_A^r, v_B^r;\right.$$
$$\left.u_A^1, u_B^1, \ldots, u_A^r, u_B^r; \max\left\{\sum_{j=1}^{r+i+1} a_j + \sum_{k=1}^{r}(v_A^k - u_A^k + \delta^k), x\right\} + b_{r+i+2}\right).$$

Given that most of the required partial sums can be computed in advance, the makespan of the partial schedule above can be updated in no more than $O(q)$ time, which is constant.

At the initialization stage we define

$$(0; v_A^1, v_B^1, \ldots, v_A^r, v_B^r;$$
$$0, 0, v_A^1 + \delta^1 + a_1, v_B^1 + b_1, \ldots, v_A^r + \delta^r + a_r, v_B^r + b_r; v_B^r + b_r).$$

In our analysis of the running time of the dynamic programming scheme, we assume that the values of the last state are stored as the objective function values. The number of schedules created at the initialization stage does not exceed $\sum_{r=1}^{q} \left(\binom{n}{r} r! a_{\max}^r (a(N) + b(N))^r \right) \leq \sum_{r=1}^{q} \left(\binom{n}{r} r! a(N)^r (a(N) + b(N))^r \right) = O(n^q a(N)^q (a(N) + b(N))^q)$, which is pseudopolynomial for a fixed q. For each initial schedule with $r \leq q$ crossover jobs, we derive that $0 \leq u_A^1 \leq v_A^1$, $v_A^{k-1} + a_{k-1} + \delta^{k-1} \leq u_A^k \leq v_A^k$ and $0 \leq u_B^1 \leq v_B^1$, $v_B^{k-1} + b_{k-1} \leq u_B^k \leq v_B^k$ for each $k = 2, \ldots, r$. This implies that u_A^k take no more than $a(N)$ integer values, and u_B^k take at most $a(N) + b(N)$ integer values for each $k = 1, 2, \ldots, r$. Thus, minimising the value of the makespan for each initial schedule requires at most $O(na(N)^q (a(N) + b(N))^q)$ time. Let M be $\max\{a(N), b(N)\}$ then the total processing time for the dynamic programming algorithm is $O(n^{q+1} M^{4q})$.

Thus, we have proved the following statement.

Theorem 3.1 *Problem* $F2|h(q,0), Re|C_{\max}$ *is solvable in pseudopolynomial time for any fixed* $q \geq 1$.

Furthermore, it can be demonstrated that the above approach carries over for problem $F2|h(q_A, q_B), Re|C_{\max}$ with q_A holes on machine A and q_B holes on machine B, where q_A and q_B are fixed. Lemma 3.1 can be extended for this general problem in the following way. Let (J', σ, J'') be any fragment of the job sequence that is associated with an optimal schedule, where the jobs J' and J'' are two consecutive crossover jobs, not necessarily interrupted by the

holes on the same machine. Then the sequence (J', σ) follows Johnson's rule. It is clear that in the optimal sequence the jobs scheduled before the first crossover job and those placed after the last crossover job also obey Johnson's rule. This property can be used for developing a pseudopolynomial dynamic programming algorithm for problem $F2|h(q_A, q_B), Re|C_{\max}$. Here, we refrain from presenting its formal description which may appear quite technical. It suffices to state its main steps. First, select $r_A \leq q_A$ potential crossover jobs on machine A and $r_B \leq q_B$ crossover jobs on machine B (a job can be a crossover job on each machine). Then, to complete the initialization stage, generate a permutation of the selected jobs and assign them into intervals so that each crossover job is interrupted by a hole on the corresponding machine. All permutations and all possible starting times of the crossover jobs have to be enumerated. In the main stage of the algorithm, starting from one of the initial schedules we assign the remaining jobs taken in a Johnson order to be processed in the gaps of the current partial schedule.

Corollary 3.1 *Problem* $F2|h(q_A, q_B), Re|C_{\max}$ *is solvable in pseudopolynomial time for any fixed* $q_A \geq 0$ *and* $q_B \geq 0$.

In the following section we discuss the conversion of the dynamic programming algorithm into an FPTAS.

3.3 Resumable Scenario: FPTAS

In this section we present fully polynomial approximation schemes (FPTAS) for the two-machine flow shop problem under the resumable scenario. We start with converting the pseudopolynomial dynamic programming algorithm for problem $F2|h(0,1), Re|C_{\max}$ designed by Lee [89] into a FPTAS for this problem. Then we use this FPTAS as a subroutine to design a FPTAS for problem $F2|h(1,0), Re|C_{\max}$. Untill recently that the best approximation algorithm has been a $\frac{4}{3}-$approximation algorithm applicable to problems $F2|h(1,0), Re|C_{\max}$ and $F2|h(0,1), Re|C_{\max}$, see Lee [90] and Cheng

and Wang [32], respectively. Ng and Kovalyov [107] propose FPTASs for these problems.

Consider problem $F2|h(0,1), Re|C_{\max}$ with the hole $[s,t]$ of length $\Delta = t - s$ on machine B and the processing times a_j and b_j. We refer to this problem as Problem P. To develop a FPTAS we use the well–known rounding technique. Given an instance of Problem P and an $\varepsilon > 0$, define $\delta = \varepsilon \max\{a(N), b(N)\}/(n+2)$.

Introduce Problem $\widetilde{P}$ as problem $F2|h(0,1), Re|C_{\max}$ with the processing times defined as

$$\widetilde{a}_j = \lfloor a_j/\delta \rfloor, \widetilde{b}_j = \lfloor b_j/\delta \rfloor, j = 1, 2, \ldots, n, \tag{3.1}$$

and the hole $\left[\widetilde{s}, \widetilde{t}\right]$ of length $\widetilde{\Delta}$, where

$$\widetilde{s} = \lfloor s/\delta \rfloor, \ \widetilde{\Delta} = \lfloor \Delta/\delta \rfloor, \ \widetilde{t} = \widetilde{s} + \widetilde{\Delta}. \tag{3.2}$$

Here $\lfloor x \rfloor$ denotes the largest integer that does not exceed x.

Algorithm FPReB

INPUT: An instance of Problem P and $\varepsilon > 0$.

OUTPUT: A heuristic schedule S_ε.

1. Define the instance of Problem $\widetilde{P}$ by (3.1) and (3.2).

2. For Problem $\widetilde{P}$, run the dynamic programming algorithm by Lee [89]. Call the found schedule and the associated permutation of job indices by $\widetilde{S}$ and π, respectively.

3. Process the jobs from the original instance of Problem P according to the permutation π, provided that each operation starts as early as possible, interrupting one of the operations on machine B, when required. Call the resulting schedule S_ε. Stop.

Theorem 3.2 *For schedule S_ε found by Algorithm FPReB the inequality*

$$\frac{C_{\max}(S_\varepsilon)}{C_{\max}(S^*)} \leq 1+\varepsilon$$

holds. The running time of the algorithm does not exceed $O\left(n^5/\varepsilon^4\right)$.

Proof. Given an instance of Problem P, introduce new auxiliary problem $F2|h(0,1), Re|C_{\max}$ with the processing times $\overline{a}_j$ and $\overline{b}_j$ defined as

$$\overline{a}_j = \delta\widetilde{a}_j, \overline{b}_j = \delta\widetilde{b}_j,\ j = 1,2,\ldots,n, \tag{3.3}$$

and the hole $\left[\overline{s},\overline{t}\right]$ such that

$$\overline{s} = \delta\widetilde{s},\ \overline{\Delta} = \delta\widetilde{\Delta},\ \overline{t} = \overline{s} + \overline{\Delta}, \tag{3.4}$$

and call this Problem $\overline{P}$.

Let π be a permutation of job indices that defines schedule $\widetilde{S}$ found in Step 2 of Algorithm FPReB. Due to (3.3) and (3.4) we derive that a schedule $\overline{S}$ that is optimal for Problem $\overline{P}$ is also associated with the same permutation. Without loss of generality, we assume that the jobs are renumbered in such a way that $\pi = (1,2,\ldots,n)$. Suppose that job J_u is critical in schedule $\overline{S}$. For schedule S_ε for the original Problem P, we assume that job J_v is critical. Recall that the processing times a_j for Problem P are obtained by extending the times $\overline{a}_j$ to their original values by no more than δ each. The same holds for other time parameters in these two problems.

If in schedule $\overline{S}$ job J_u either is processed on machine B before the hole or is the crossover job in that schedule, then

$$C_{\max}(\overline{S}) = \sum_{j=1}^{u}\overline{a}_j + \sum_{j=u}^{n}\overline{b}_j + \overline{\Delta}.$$

Irrespective of the position of job J_v in schedule S_ε we have that

$$C_{\max}(S_\varepsilon) \leq \sum_{j=1}^{v} a_j + \sum_{j=v}^{n} b_j + \Delta,$$

which implies that

$$\begin{aligned} C_{\max}(S_\varepsilon) &\leq \sum_{j=1}^{v}(\overline{a}_j+\delta)+\sum_{j=v}^{n}(\overline{b}_j+\delta)+(\overline{\Delta}+\delta) \\ &\leq \sum_{j=1}^{u}\overline{a}_j+\sum_{j=u}^{n}\overline{b}_j+\overline{\Delta}+(n+2)\delta = C_{\max}(\overline{S})+(n+2)\delta. \end{aligned}$$

Suppose now that in schedule $\overline{S}$ job J_u starts on machine B after the hole, i.e.,

$$C_{\max}(\overline{S}) = \sum_{j=1}^{u}\overline{a}_j+\sum_{j=u}^{n}\overline{b}_j.$$

If job J_v in schedule S_ε also starts on B after the hole, we have that

$$C_{\max}(S_\varepsilon) = \sum_{j=1}^{v}a_j+\sum_{j=v}^{n}b_j,$$

which implies

$$\begin{aligned} C_{\max}(S_\varepsilon) &\leq \sum_{j=1}^{v}(\overline{a}_j+\delta)+\sum_{j=v}^{n}(\overline{b}_j+\delta) \\ &\leq \sum_{j=1}^{u}\overline{a}_j+\sum_{j=u}^{n}\overline{b}_j+(n+1)\delta \leq C_{\max}(\overline{S})+(n+2)\delta. \end{aligned}$$

Finally, consider the situation that in schedule S_ε the critical job J_v starts on B before the hole, i.e.,

$$C_{\max}(S_\varepsilon) = \sum_{j=1}^{v}a_j+\sum_{j=v}^{n}b_j+\Delta.$$

Since in schedule $\overline{S}$ job J_u is critical, the inequality

$$\sum_{j=v}^{u-1}\overline{b}_j+\overline{\Delta} \leq \sum_{j=v+1}^{u}\overline{a}_j \tag{3.5}$$

obviously holds, provided that in $\overline{S}$ job J_v starts before the hole. We show that even if job J_v starts after the hole in schedule $\overline{S}$, inequality (3.5) is still

valid. To see this, it suffices to prove that in schedule $\overline{S}$ job J_v completes on A no later than time $\overline{s}$. Suppose that job J_v completes on A later than time $\overline{s}$. Recall that the transition of schedule $\overline{S}$ into schedule S_ε will shift the hole to the right by strictly less than δ time units to provide a gap for job J_v to start. On the other hand, since in Problem $\overline{P}$ all time parameters are multiples of δ, it follows that job J_v completes on A later than time $\overline{s}$ by at least δ, so that in schedule S_ε this job cannot start on B before the hole, a contradiction.

Using inequality (3.5), we derive

$$\begin{aligned}
C_{\max}(S_\varepsilon) &= \sum_{j=1}^{v} a_j + \sum_{j=v}^{u-1} b_j + \Delta + \sum_{j=u}^{n} b_j \\
&\leq \sum_{j=1}^{v} a_j + \left(\sum_{j=v}^{u-1} \overline{b}_j + \overline{\Delta} + (u-v+1)\delta \right) + \sum_{j=u}^{n} b_j \\
&\leq \left(\sum_{j=1}^{v} \overline{a}_j + v\delta \right) + \left(\sum_{j=v+1}^{u} \overline{a}_j + (u-v+1)\delta \right) \\
&\quad + \left(\sum_{j=u}^{n} \overline{b}_j + (n-u+1)\delta \right) \\
&\leq \left(\sum_{j=1}^{u} \overline{a}_j + \sum_{j=u}^{n} \overline{b}_j \right) + (n+2)\delta = C_{\max}(\overline{S}) + (n+2)\delta.
\end{aligned}$$

This due to the definition of δ yields

$$C_{\max}(S_\varepsilon) \leq C_{\max}(\overline{S}) + \varepsilon \max\{a(N), b(N)\} \leq (1+\varepsilon) C_{\max}(S^*).$$

The running time of Algorithm FPReB is determined by the running time of the dynamic programming algorithm used in Step 2, that requires $O(n(\widetilde{a}(N)+\widetilde{b}(N))^2 \widetilde{s} \max \widetilde{b}_j) = O(n(\max\{\widetilde{a}(N), \widetilde{b}(N)\})^4)$ time. The definition of $\widetilde{a}_j$ implies that $\max\{\widetilde{a}(N), \widetilde{b}(N)\} \leq (n+2)/\varepsilon$. Thus, we conclude that the running time of Algorithm FPReB does not exceed $O(n^5/\varepsilon^4)$, and the algorithm is a fully polynomial approximation scheme. ■

Now we focus on problem $F2|h(1,0), Re|C_{\max}$ with the hole $[s,t]$ of length Δ on machine A and the job processing times a_j and b_j. We refer to this

problem as Problem Q. Notice that this problem is not a mirror image of problem $F2|h(0,1), Re|C_{\max}$ considered earlier in this section. Let S_Q be a feasible schedule for Problem Q, where $C_{\max}(S_Q) = \lambda$. Associate with Problem Q an instance of problem $F2|h(0,1), Re|C_{\max}$ in which the processing time of each job J_j on machine A is equal to b_j, while its processing time on machine B is equal to a_j. Define the hole $[s', t']$ on machine B by setting $s' = \lambda - s$ and $t' = s' - \Delta$. We refer to this problem as Problem $P(\lambda)$. If in Problem $P(\lambda)$ the jobs are processed in the reverse order to that in schedule S_Q, we obtain a schedule $S_{P(\lambda)}$ that is feasible for Problem $P(\lambda)$ and such that $C_{\max}(S_Q) = C_{\max}(S_{P(\lambda)}) = \lambda$.

In what follows, we assume that $a(N) > s$, i.e., in Problem Q all jobs cannot be completed on machine A before the hole, so that $a(N)+\Delta$ is a lower bound on the optimal makespan. If $a(N) \leq s$, Problem Q is polynomially solvable.

In order to find an approximate solution to the original Problem Q, we use Algorithm FPReB for finding an approximate solution to Problem $P(\lambda)$ with an appropriate λ. The value of λ is chosen by binary search.

For the original Problem Q, given an $\varepsilon > 0$, we will apply Algorithm FPReB to Problem $P(\lambda)$ with $\varepsilon' = \varepsilon/2$. We call an application of Algorithm FPReB to Problem $P(\lambda)$ *successful* if for the found schedule S_ε we have that $C_{\max}(S_\varepsilon) \leq \lambda$ (and λ can be reduced), and *unsuccessful* otherwise (in which case λ has to be enlarged).

In the beginning of the process, define upper and lower bounds on an optimal makespan:

$$\overline{\lambda} = UB := a(N) + b(N) + \Delta, \ \underline{\lambda} = LB := \max\{a(N) + \Delta, b(N)\}.$$

Define $\lambda := (\overline{\lambda} + \underline{\lambda})/2$, form the instance of Problem $P(\lambda)$ and run Algorithm FPReB. If its application is successful redefine $\overline{\lambda} := \lambda$; otherwise, redefine $\underline{\lambda} := \lambda$. Stop if $\overline{\lambda} - \underline{\lambda} \leq \varepsilon' \max\{a(N) + \Delta, b(N)\}$; otherwise set $\lambda := (\overline{\lambda} + \underline{\lambda})/2$ and repeat the process again. Upon completion of the

process, denote the schedule found for Problem $P(\overline{\lambda})$ by $S_{P(\overline{\lambda})}$ and convert it into schedule S_Q for the original problem by inverting the job sequence and starting each operations as early as possible. It follows that

$$\begin{aligned} C_{\max}(S_Q) &\leq (\overline{\lambda} - \underline{\lambda}) + C_{\max}(S_{P(\overline{\lambda})}) \\ &\leq \varepsilon' \max\{a(N) + \Delta, b(N)\} + (1+\varepsilon')C_{\max}(S^*) \\ &\leq (1+\varepsilon)C_{\max}(S^*), \end{aligned}$$

as required. During this process Algorithm FPReB is called $O(\log(UB-LB))$ times, hence we obtain a FPTAS for the original Problem Q.

3.4 Resumable Scenario: $\frac{3}{2}-$ Approximation

As seen from the previous sections, for problem $F2|h(q,0), Re|C_{\max}$ the running times of the dynamic programming algorithm and that of the FPTAS (for $q = 1$) are high. It is therefore reasonable to try to develop a fast heuristic algorithm that guarantees a solution fairly close to the optimum. For this problem with only one hole on machine A, Lee [89] presents a $\frac{3}{2}-$approximation algorithm. We extend his heuristic to the case of several holes and simplify both the algorithm and the proof. Notice that no approximation algorithms for the flow shop problems with more than one hole have been available so far.

Our algorithm creates two schedules and outputs the better of them as a heuristic solution.

Algorithm HqA

INPUT: An instance of Problem $F2|h(q,0), Re|C_{\max}$.

OUTPUT: A heuristic schedule S_H.

1. Select the job with the largest processing time on machine B and place it into the first position in the processing sequence, followed by all other jobs in an arbitrary order. Call the schedule associated with that sequence by S_1.

2. Order all jobs in non-increasing order of b_j/a_j and denote the schedule associated with that sequence by S_2.

3. Among schedules S_1 and S_2 select the one with the smaller makespan and output it as a heuristic solution S_H.

Since Step 2 of Algorithm HqA requires sorting the set of jobs N its overall running time is $O(n \log n)$. We now analyze its worst-case performance and prove that the inequality

$$\frac{C_{\max}(S_H)}{C_{\max}(S^*)} \leq \frac{3}{2} \tag{3.6}$$

holds for any instance of problem $F2|h(q,0), Re|C_{\max}$, where S^* is an optimal schedule.

Our consideration is based on the following statement.

Lemma 3.2 *Suppose we schedule jobs in non-increasing order of b_j/a_j and obtain schedule S_2 for problem $F2|h(q,0), Re|C_{\max}$. Let J_k be the critical job of the schedule then we have*

$$C_{\max}(S_2) - C_{\max}(S^*) \leq b_k.$$

This lemma has been proved by Lee [89] for the problem with a single hole. It is easy to extend his argument to the case of several holes, since the proof uses no information regarding the lengths of the holes or their number.

Theorem 3.3 *Let S_H be the schedule found by Algorithm HqA for problem $F2|h(q,0), Re|C_{\max}$. Then the bound (3.6) holds, and this bound is tight.*

Proof. We are interested only in the case that $a(N) > s_1$; otherwise Johnson's algorithm delivers an optimal solution in polynomial time.

Suppose that there is a job with the processing time on machine B greater than $\frac{1}{2}C_{\max}(S^*)$. Without loss of generality, call this job J_1. Since $b_1 > \frac{1}{2}C_{\max}(S^*)$, we have that

$$\sum_{j=2}^{n} b_j < \frac{1}{2}C_{\max}(S^*). \tag{3.7}$$

In schedule S_1 found in Step 1 of the algorithm, job J_1 is processed first. We may obtain two possible situations regarding the placement of the critical job in this schedule.

Job J_1 is critical. Since J_1 is scheduled first, its completion time on machine B is a lower bound on the makespan of an optimal schedule, so that $C_{\max}(S_1) \leq C_{\max}(S^*) + \sum_{j=2}^{n} b_j$. Due to (3.7), we derive that (3.6) holds for $S_H = S_1$.

Job J_1 is not critical. Since the completion time of the critical job on machine A is a lower bound on the optimal makespan, we again obtain $C_{\max}(S_1) \leq C_{\max}(S^*) + \sum_{j=2}^{n} b_j$, so that the theorem holds.

In the remaining case that $b_j < \frac{1}{2}C_{\max}(S^*)$ for all $j = 1, 2, \ldots, n$, we apply Lemma 3.2 to obtain $C_{\max}(S_2) \leq C_{\max}(S^*) + b_k$, which in turn implies

$$\frac{C_{\max}(S_2)}{C_{\max}(S^*)} \leq 1 + \frac{b_k}{C_{\max}(S^*)} \leq \frac{3}{2}.$$

To see that the bound (3.6) is tight, consider the following instance of problem $F2|h(q,0), Re|C_{\max}$. There are two jobs such that $a_1 = k+1, b_1 = k^2+3k+2$ and $a_2 = k, b_2 = k^2+k+1$, where k is an integer greater than 1. The hole on machine A occupies the interval $[k, k^2+k]$. Since $b_1 = k^2+3k+2 > k^2+k+1 = b_2$, it follows that in Step 1 of the algorithm we obtain schedule S_1 associated with the sequence (J_1, J_2). Since for $k > 1$ we have that

$$\frac{b_1}{a_1} = \frac{k^2+3k+2}{k+1} = k+2 > \frac{k^2+k+1}{k} = \frac{b_2}{a_2},$$

it follows that in Step 2 we obtain schedule S_2 associated with the same sequence. It is easy to verify that $C_{\max}(S_H) = 3k^2+5k+4$. On the other hand, for the optimal schedule S^* the sequence of jobs is (J_2, J_1), so that we have $C_{\max}(S^*) = 2k^2+5k+3$. Thus, as k approaches infinity $C(S_H)/C(S^*)$ goes to 3/2. ■

3.5 Semi-Resumable Scenario: PTAS

In this section we study the two-machine flow shop problem with a single hole under the semi-resumable scenario. We concentrate on the case of the hole on machine B, i.e., consider problem $F2|h(0,1), S\text{-}Re|C_{\max}$. The case of the hole on the other machine is symmetric. Recall that under the semi-resumable scenario, the operation of the crossover job J_k has to be partially redone, and $\alpha_k \in [0,1]$ is a given parameter that determines the size of the required reprocessing.

For each problem $F2|h(0,1), S\text{-}Re|C_{\max}$ and $F2|h(1,0), S\text{-}Re|C_{\max}$, we offer a PTAS. Notice that the best results in this area available so far are a $\frac{3}{2}-$approximation algorithm for problem $F2|h(0,1), S\text{-}Re|C_{\max}$ and a $2-$approximation algorithm for problem $F2|h(1,0), S\text{-}Re|C_{\max}$, see Lee [90].

Our PTAS has the following features. First, we follow the useful idea of Sevastianov and Woeginger [134] of splitting the jobs into big, medium and small. We look for an approximate solution in one of three classes of schedules, depending on the position and the size of the crossover job. For each of these classes we enumerate all schedules of the big jobs and try to schedule the small jobs in the gaps of that schedule by solving an integer programming problem (or rather its linear programming relaxation). Those jobs that cannot be fully processed in the existing gaps (including all medium jobs) are appended.

We first specify how the big, medium and small jobs are defined. Let $T = a(N) + b(N)$. Consider an arbitrary ε, where $0 < \varepsilon < 1$. We define $\tilde{\varepsilon} = \varepsilon/10$ and introduce the sequence of real numbers $\delta_1, \delta_2, \ldots$, where $\delta_t = \tilde{\varepsilon}^{2^t}$.

For each integer t, where $t \geq 1$, consider the set of jobs

$$N^t = \{J_j | j \in N,\ \delta_t^2 T < a_j + b_j \leq \delta_t T\}.$$

Note that the sets of jobs $N^1, N^2, \ldots$ are mutually disjoint. Thus, there exists a $\tau \in \{1, \ldots, \lceil 1/\tilde{\varepsilon} \rceil\}$ such that $p(N^\tau) \leq \tilde{\varepsilon} T$ holds, where $p(N^\tau)$ is

the makespan of the Johnson sequence of the jobs of set N^τ; otherwise $T \geq p(N^1) + \cdots + p(N^{\lceil \frac{1}{\tilde{\varepsilon}} \rceil}) > \lceil \frac{1}{\tilde{\varepsilon}} \rceil \tilde{\varepsilon} T$, which is impossible. We define $\delta = \delta_\tau$, and note that

$$\tilde{\varepsilon}^{2^{\lceil 1/\tilde{\varepsilon} \rceil}} \leq \delta \leq \tilde{\varepsilon}^2.$$

We now partition the jobs into *big jobs* W_b, *medium jobs* W_m and *small jobs* W_s by partitioning the index set N as follows:

$$\begin{aligned} W_b &= \{j | a_j + b_j > \delta T\}, \\ W_m &= \{j | \delta^2 T < a_j + b_j \leq \delta T\}, \\ W_s &= \{j | a_j + b_j \leq \delta^2 T\}. \end{aligned} \tag{3.8}$$

Note that, by definition, $W_m = N^\tau$, which implies

$$p(W_m) \leq \tilde{\varepsilon} T. \tag{3.9}$$

Hence complementing the partial schedule for the big and small jobs with a Johnson's sequence of the medium jobs will increase the makespan by at most $\tilde{\varepsilon} T$.

Let n_b denote the number of big jobs. From the definition of W_b, each big job has a total processing time that exceeds δT. Since the total processing time of all jobs is equal to T, we deduce that $n_b \leq 1/\delta$. Moreover, $1/\delta \leq \tilde{\varepsilon}^{-2^{\lceil 1/\tilde{\varepsilon} \rceil}}$, which implies that n_b is fixed.

Our approximation scheme involves searching for an approximate solution in several specific classes of schedules. For notational convenience, we denote the big jobs by J'_k for $k = 1, \ldots, n_b$, and their processing times on machines A and B by a'_k and b'_k, respectively. Define

$\mathcal{S}_0$ - the class of schedules in which all big jobs are completed before the hole;

$\bar{\mathcal{S}}_v$ - the class of schedules in which no big job is interrupted by the hole and a big job J'_v is the first big job that starts on B after the hole;

$\tilde{\mathcal{S}}_v$ - the class of schedules in which a big job J'_v is interrupted by the hole.

It is clear that an optimal schedule S^* belongs to one of these classes.

Introduce two dummy jobs J'_0 and J'_{n_b+1}, where each dummy job has a zero processing time on both machines. These jobs are needed for simplification of a linear programming problem that will be a part of our algorithm.

Case 1.

We first look for an approximate solution in class $\mathcal{S}_0$. In a schedule of this class the crossover job is not a big job and all big jobs are sequenced by Johnson's rule. If necessary, renumber the big jobs in such a way that a Johnson sequence of these jobs is given by $(J'_0, J'_1, \ldots, J'_{n_b}, J'_{n_b+1})$ with the two dummy jobs at the beginning and at the rear of the sequence.

Let n_s denote the number of small jobs and denote the small jobs by $J_1, \ldots, J_{n_s}$. We define variables

$$x_{jk} = \begin{cases} 1, & \text{if } J_j \text{ is scheduled between jobs } J'_{k-1} \text{ and } J'_k, \\ 0, & \text{otherwise,} \end{cases}$$

for $j \in W_s$ and $k = 1, \ldots, n_b+1$. The following integer program is a relaxation of the problem of finding a schedule from class $\mathcal{S}_0$ for processing the big jobs and the small jobs. The variable C provides a lower bound on the makespan of that partial schedule. We call this integer program $IP(0)$.

$$C \to \min$$

$$\text{s.t.} \sum_{k=1}^{u} \left(\sum_{j \in W_s} a_j x_{jk} + a'_k \right) + b'_u + \sum_{k=u+1}^{n_b+1} \left(\sum_{j \in W_s} b_j x_{jk} + b'_k \right) \le C,$$

$$u = 1, \ldots, n_b + 1; \tag{3.10}$$

$$\sum_{k=1}^{u} \left(\sum_{j \in W_s} a_j x_{jk} + a'_k \right) + b'_u + \sum_{k=u+1}^{n_b} \left(\sum_{j \in W_s} b_j x_{jk} + b'_k \right) \le s, \quad u = 1, \ldots, n_b; \tag{3.11}$$

$$\sum_{k=1}^{n_b+1} x_{jk} = 1, \qquad j \in W_s; \tag{3.12}$$

$$x_{jk} \in \{0, 1\}, \qquad j \in W_s, \ k = 1, \ldots, n_b + 1. \tag{3.13}$$

Constraints (3.10) give lower bounds on the makespan, provided that big job J'_u is critical. Constraints (3.11) imply that all big jobs in the partial

schedule must be completed on B before the hole. Constraints (3.12) reflect the fact that each small job must be sequenced between some pair of big jobs, including the dummy jobs.

We solve the linear programming relaxation of this problem in which the constraints $x_{ij} \in \{0,1\}$ in (3.13) are replaced by the non-negativity constraints $x_{ij} \geq 0$. Any small job J_j for which $x_{jk} \neq 1$ for any position k in this solution is called a *fractional* job. Note that there are $2n_b + n_s + 1$ constraints, and consequently $2n_b + n_s + 1$ basic variables, including C which must be basic. Moreover, each of the n_s assignment constraints (3.12) contains a distinct set of variables. Following the same type of analysis as that of Potts [124], we establish that there are at most $2n_b$ fractional jobs.

Replace each fractional job J_j with several pseudo-jobs J_j^k for all k such that $x_{jk} > 0$. A pseudo-job J_j^k is assigned to a position between jobs J'_{k-1} and J'_k, and its processing times on machines A and B are set equal to $a_j^k = a_j x_{jk}$ and $b_j^k = b_j x_{jk}$. Each non-fractional small job J_j with $x_{jk} = 1$ is assigned to a position between jobs J'_{k-1} and J'_k. For $k = 1, \ldots, n_b + 1$, the small non-fractional jobs and pseudo-jobs assigned to positions between jobs J'_{k-1} and J'_k are sequenced according to Johnson's rule.

Let S^0_{LP} be a schedule associated with the found permutation, provided that the crossover job, if exists, is processed in accordance with the chosen scenario. Remove all pseudo-jobs and assign all fractional small jobs and all medium jobs to be processed in an arbitrary order in such a way the first of these jobs starts on machine A at time $\max\{C_{\max}(S^0_{LP}), t\}$. Call the resulting schedule S_ε.

We prove that

$$C_{\max}(S_\varepsilon) \leq C_{\max}(S^*) + (2n_b + 2)\delta^2 T + \tilde{\varepsilon} T, \tag{3.14}$$

provided that S^* is an optimal schedule that also belongs to class $\mathcal{S}_0$. Let C^0 denote an optimal value of C in the linear programming relaxation of the integer program $IP(0)$. Let J_r be a crossover job in schedule S^0_{LP}; recall that

in this case J_r is either a small job or a pseudo-job. Recall that according to the semi-resumable scenario job J_r will be reprocessed on machine B for no more than $\alpha_r b_r$ extra time units.

If in schedule S^0_{LP} a big job J'_u is critical, then the value of $C_{\max}(S^0_{LP})$ exceeds that of C^0 by no more than $\alpha_r b_r$. Thus, $C_{\max}(S^0_{LP}) \leq C^0 + \delta^2 T \leq C_{\max}(S^*) + \delta^2 T$.

Suppose that in schedule S^0_{LP} the critical job belongs to the set W_k of small jobs and pseudo-jobs positioned between the big jobs J'_{k-1} and J'_k for some $k, 1 \leq k \leq n_b + 1$. Without loss of generality, we may assume that the critical job us a non-fractional small job J_w; the case of a pseudo-job being critical is analogous. Consider the contribution to the value of $C_{\max}(S^0_{LP})$ that is delivered by the jobs in W_s. Job J_w contributes $a_w + b_w$, a job J_j (or pseudo-job J^k_j) that precedes J_w contributes $a_j x_{jk}$, while a job J_j (or pseudo-job J^k_j) that follows J_w contributes $b_j x_{jk}$. If $a_w \leq b_w$ then according to Johnson's rule we have that $a_j x_{jk} \leq b_j x_{jk}$ for all jobs of W_s that precede J_w; similarly, if $a_w > b_w$ then $a_j x_{jk} > b_j x_{jk}$ for all jobs of W_s that follow J_w. In any case, the contribution of the jobs that are contained in set W_s to the makespan $C_{\max}(S^0_{LP})$ does not exceed $a_w + b_w$ plus total processing time of these jobs on one of the machines. This implies that the length of the critical path with job J_w being critical does not exceed the length of the longer path is which either big job J'_{k-1} or big job J'_k is critical plus the value of $a_w + b_w$. As above, the length of the critical path with a big critical job is bounded by $C^0 + \delta^2 T$. Thus, if a big job is not critical in schedule S^0_{LP}, we derive that $C_{\max}(S^0_{LP}) \leq C^0 + 2\delta^2 T \leq C_{\max}(S^*) + 2\delta^2 T$.

When the pseudo-jobs are removed from schedule S^0_{LP} and the fractional jobs and the medium jobs are appended to that schedule, for the resulting schedule we have that

$$C_{\max}(S_\varepsilon) \leq \max\{C_{\max}(S^0_{LP}), t\} + 2n_b\delta^2 T + \tilde{\varepsilon} T \leq C_{\max}(S^*) + (2n_b + 2)\delta^2 T + \tilde{\varepsilon} T.$$

Case 2.

We now look for an approximate solution such that a big job J'_v, where $1 \leq v \leq n_b$, is the first big job that completes on machine B after the hole. As shown by Lee [89, 90], we only have to consider schedules in which the big jobs that precede job J'_v are sequenced by Johnson's rule, and so are the big jobs that follow job J'_v. Thus, to obtain a sequence of big jobs that is associated with a certain schedule we need to split the set of the remaining big jobs into two subsets, so that the jobs of one subset are positioned before job J'_v (we call this subset the *front part*) and the jobs of the other subset are positioned after job J'_v (we call this subset the *rear part*). Our approximation scheme will generate all possible partitions of the set of big jobs into the front and rear parts.

We give further description and analysis of the approximation scheme, provided that job J'_v is fixed and the partition of the remaining big jobs into the front and rear parts is also fixed. Suppose that there are $h-1$ big jobs in the front part. Sequence the jobs in the front part and those in the rear part by Johnson's rule. For notational convenience, relabel the big jobs in such a way that the obtained sequence is given by $(I_0, I_1, \ldots, I_{h-1}, I_h = J'_v, I_{h+1}, \ldots, I_{n_b}, I_{n_b+1})$, where I_0 and I_{n_b+1} are the dummy jobs with zero processing times on both machines (similar to J'_0 and J'_{n_b+1} used in Case 1). Similarly to Case 1, for a big job I_j denote its processing times on machines A and B by a'_j and b'_j, respectively.

For a small job J_j, we define the variables

$$x_{jk} = \begin{cases} 1, & \text{if } J_j \text{ is scheduled between jobs } I_{k-1} \text{ and } I_k, \\ 0, & \text{otherwise,} \end{cases}$$

for $k = 1, \ldots, n_b + 1$.

Case 2a.

We first study the situation that an approximate solution to the original problem is sought for in class $\bar{\mathcal{S}}_v$. The following integer program is a relaxation of the problem of finding a schedule from class $\bar{\mathcal{S}}_v$ for processing

the big jobs and the small jobs. The variable C provides a lower bound on the makespan of that partial schedule. The variable R_h corresponds to the starting time of job $I_h = J'_v$ on machine B. We call this integer program $\overline{IP}(v)$.

$$C \rightarrow \min$$

$$\text{s.t. } t \leq R_h; \tag{3.15}$$

$$\sum_{k=1}^{h-1} \left(\sum_{j \in W_s} a_j x_{jk} + a'_k \right) + a'_h \leq R_h; \tag{3.16}$$

$$\sum_{k=1}^{u} \left(\sum_{j \in W_s} a_j x_{jk} + a'_k \right) + b'_u + \sum_{k=u+1}^{h} \left(\sum_{j \in W_s} b_j x_{jk} + b'_k \right) - b'_h + \Delta \leq R_h,$$

$$u = 1, \ldots, h-1; \tag{3.17}$$

$$R_h + b'_h + \sum_{k=h+1}^{n_b+1} \left(\sum_{j \in W_s} b_j x_{jk} + b'_k \right) \leq C; \tag{3.18}$$

$$\sum_{k=1}^{u} \left(\sum_{j \in W_s} a_j x_{jk} + a'_k \right) + b'_u + \sum_{k=u+1}^{n_b+1} \left(\sum_{j \in W_s} b_j x_{jk} + b'_k \right) \leq C;$$

$$u = h+1, \ldots, n_b; \tag{3.19}$$

$$\sum_{k=1}^{u} \left(\sum_{j \in W_s} a_j x_{jk} + a'_k \right) + b'_u + \sum_{k=u+1}^{h-1} \left(\sum_{j \in W_s} b_j x_{jk} + b'_k \right) \leq s;$$

$$u = 1, \ldots, h-1; \tag{3.20}$$

$$\sum_{k=1}^{n_b+1} x_{jk} = 1, \qquad j \in W_s; \tag{3.21}$$

$$x_{jk} \in \{0, 1\}, \qquad j \in W_s,\ k = 1, \ldots, n_b + 1. \tag{3.22}$$

Constraints (3.15) and (3.17) guarantee that job I_h starts on B after the hole and not earlier than a preceding small job completes on that machine, provided that big job I_u is critical, $1 \leq u \leq h-1$. Constraint (3.16) does not allow job I_h to start on B earlier than that job completes on machine A. Constraints (3.18) and (3.19) give lower bounds on the value of makespan, provided that job I_u is critical. Constraint (3.20) implies that all big jobs

$I_1, \ldots, I_{h-1}$ in the partial schedule must be completed on B before the hole. Constraints (3.21) have the same meaning as in $IP(0)$. Notice that in the formulation of this integer program the resumable scenario is assumed.

As in Case 1, we solve the linear programming relaxation of this problem in which the constraints $x_{ij} \in \{0,1\}$ in (3.22) are replaced by the non-negativity constraints $x_{ij} \geq 0$. The relaxation problem may appear to be infeasible, but that only means that a wrong partition has been used for a given job J'_v; for further purposes we are only interested in situations that a linear programming relaxation can be solved to optimality. Similarly to Case 1, it can be verified that a basic optimal solution of the relaxation problem contains at most $n_b + h - 1$ fractional jobs, which is again no more than $2n_b$.

The resulting schedule S_ε can be found as in Case 1, i.e., by introducing pseudo-jobs; ordering the jobs between the big jobs according to Johnson's rule; determining a schedule $\bar{S}^v_{LP}$ associated with the found permutation, provided that the crossover job, if exists, is processed in accordance with the chosen scenario; removing all pseudo-jobs and appending all fractional small jobs and all medium jobs in an arbitrary order starting at time $C_{\max}(\bar{S}^v_{LP})$.

We prove that (3.14) holds, provided that S^* is an optimal schedule associated with the same choice of job $J'_v = I_h$ and the same partition of the other big jobs into the front and rear parts. Let $\bar{C}^v$ denote an optimal value of C in the linear programming relaxation of the integer program $\overline{IP}(v)$.

Similarly to Case 1, it can be seen that

- $C_{\max}(\bar{S}^v_{LP}) = \bar{C}^v$ if in schedule $\bar{S}^v_{LP}$ a big job J'_u for $u \geq h$ is critical; or
- $C_{\max}(\bar{S}^v_{LP}) \leq \bar{C}^v + \delta^2 T$ if either the critical job is either a big job J'_u for $u \leq h-1$ or one of the small jobs (or, possibly, pseudo-jobs) positioned after job I_h; or
- $C_{\max}(\bar{S}^v_{LP}) \leq \bar{C}^v + 2\delta^2 T$ for any other critical job.

When the pseudo-jobs are removed from schedule $\bar{S}^v_{LP}$ and the fractional

jobs and the medium jobs are appended to that schedule, for the resulting schedule we have that

$$C_{\max}(S_\varepsilon) \leq C_{\max}(\bar{S}^v_{LP}) + 2n_b\delta^2 T + \tilde{\varepsilon}T \leq C_{\max}(S^*) + (2n_b + 2)\delta^2 T + \tilde{\varepsilon}T.$$

Case 2b.

We now consider the situation that an approximate solution to the original is sought for in class $\tilde{\mathcal{S}}_v$. The following integer program is a relaxation of the problem of finding a schedule from class $\tilde{\mathcal{S}}_v$ for processing the big jobs and the small jobs. The variable C provides a lower bound on the makespan of that partial schedule. The variable R_h corresponds to the starting time of job $I_h = J'_v$ on machine B. We call this integer program $\widetilde{IP}(v)$.

$$C \rightarrow \min$$

$$\text{s.t. } R_h \leq s; \tag{3.23}$$

$$R_h + b'_h > s; \tag{3.24}$$

$$\sum_{k=1}^{h-1}\left(\sum_{j\in W_s} a_j x_{jk} + a'_k\right) + a'_h \leq R_h; \tag{3.25}$$

$$\sum_{k=1}^{u}\left(\sum_{j\in W_s} a_j x_{jk} + a'_k\right) + b'_u + \sum_{k=u+1}^{h}\left(\sum_{j\in W_s} b_j x_{jk} + b'_k\right) - b'_h \leq R_h,$$
$$u = 1, \ldots, h-1; \tag{3.26}$$

$$R_h + b'_h + \Delta + \alpha_h(s - R_h) + \sum_{k=h+1}^{n_b+1}\left(\sum_{j\in W_s} b_j x_{jk} + b'_k\right) \leq C; \tag{3.27}$$

$$\sum_{k=1}^{u}\left(\sum_{j\in W_s} a_j x_{jk} + a'_k\right) + b'_u + \sum_{k=u+1}^{n_b+1}\left(\sum_{j\in W_s} b_j x_{jk} + b'_k\right) \leq C,$$
$$u = h+1, \ldots, n_b; \tag{3.28}$$

$$\sum_{k=1}^{n_b+1} x_{jk} = 1, \qquad j \in W_s; \tag{3.29}$$

$$x_{jk} \in \{0,1\}, \qquad j \in W_s,\ k = 1, \ldots, n_b + 1. \tag{3.30}$$

Notice that in the formulation of this integer program the semi-resumable scenario is applied. Constraints (3.23) and (3.24) imply job I_h is the crossover

job. Constraints (3.26) guarantee that job I_h starts on B not earlier than a preceding small job completes on that machine, provided that big job I_u is critical, $1 \leq u \leq h-1$. Constraint (3.25) does not allow job I_h to start on B earlier than that job completes on machine A. Constraint (3.27) gives a lower bound on the makespan, provided that job I_h is the crossover job, and one of the jobs I_u is critical, $1 \leq u \leq h$. Constraints (3.28) give lower bounds on the value of makespan, provided that job I_u is critical, $h+1 \leq u \leq n_b$.

We solve the linear programming relaxation of this problem. As in the Case 2a, ignoring the problems that appear to be infeasible, it can be verified that a basic optimal solution of the relaxation problem contains at most n_b+1 fractional jobs, which is again no more than $2n_b$.

The resulting schedule S_ε can be found as in Case 2a. Let $\tilde{S}^v_{LP}$ be an analogue of schedule $\bar{S}^v_{LP}$ defined in Case 2a. Notice that in schedule $\bar{S}^v_{LP}$ a crossover job is not a big job, while in schedule $\tilde{S}^v_{LP}$ the crossover job is job I_h that starts on B at time R_h, as determined by the solution of the linear programming relaxation.

We prove that (3.14) holds, provided that S^* is an optimal schedule associated with the same choice of the crossover job $J'_v = I_h$ and the same partition of the other big jobs into the front and rear parts. Let $\tilde{C}^v$ denote an optimal value of C in the linear programming relaxation of the integer program $\widetilde{IP}(v)$.

Similarly to the previous case, it can be seen that either $C_{\max}(\tilde{S}^v_{LP}) = \tilde{C}^v$ (if in schedule $\tilde{S}^v_{LP}$ a big job is critical); or $C_{\max}(\tilde{S}^v_{LP}) \leq \bar{C}^v + \delta^2 T$ for any other critical job.

When the pseudo-jobs are removed from schedule $\tilde{S}^v_{LP}$ and the fractional jobs and the medium jobs are appended to that schedule, for the resulting schedule we have that

$$C_{\max}(S_\varepsilon) \leq C_{\max}(\tilde{S}^v_{LP}) + (n_b+1)\delta^2 T + \tilde{\varepsilon} T \leq C_{\max}(S^*) + (2n_b+2)\delta^2 T + \tilde{\varepsilon} T.$$

The value of δ is chosen in such a way that $n_b \leq 1/\delta$ and $\delta \leq \tilde{\varepsilon}^2 \leq \tilde{\varepsilon}$, so

that for $\tilde{\varepsilon} < 1$ we have that

$$(2n_b + 2)\delta^2 \leq 2\delta + 2\delta^2 \leq 4\delta \leq 4\tilde{\varepsilon}^2 \leq 4\tilde{\varepsilon}.$$

This implies that

$$C_{\max}(S_\varepsilon) \leq C_{\max}(S^*) + 5\tilde{\varepsilon}T \leq (1+\varepsilon)C_{\max}(S^*), \tag{3.31}$$

where the final inequality is obtained from $T \leq 2C_{\max}(S^*)$ and our choice $\tilde{\varepsilon} = \varepsilon/10$.

We now provide the main result in this section.

Theorem 3.4 *For problem $F2|h(0,1), S\text{-}Re|C_{\max}$ the family of approximation algorithms for finding schedule S_ε is a polynomial-time approximation scheme.*

Proof. Inequality (3.31) establishes that some schedule S_ε that is generated by the algorithm provides a makespan that is no more than $1+\varepsilon$ times the optimal makespan. Thus, it remains to show that the algorithm requires polynomial time.

The algorithm constructs at most $n_b 2^{n_b} + 1$ schedules, one in class $\mathcal{S}_0$ and at most 2^{n_b-1} schedules in each class $\bar{\mathcal{S}}_v$ and $\tilde{\mathcal{S}}_v$ due to partitioning the big jobs other than job J'_v. This number of schedules is fixed for a fixed ε. For each schedule, a linear programming problem is solved, the number of variables and the number of constraints being bounded from above by a polynomial of n_s and n_b. Such a linear program is solvable in polynomial time, using the algorithm of Vaidya [144], for example. To obtain the final schedule from the solution of the linear program, the small jobs are sequenced using Johnson's algorithm, see Johnson [68], in $O(n_s \log n_s)$ time. Thus, the running time of the algorithm is polynomial. ■

3.6 Application of the developed method

The method of constructing a pseudopolynomial dynamic programming algorithm for the resumable flow shop probem, which has been described in the

Section 3.2, can be used not only for scheduling problems with availability constraints. Slightly modified, it can be applied to the flow shop problem with regular and no-wait jobs. This section is based on the paper [16].

We consider a version of the two-machine flow shop which is, in fact, a combination of two classical flow shop problems: $F2|\ |C_{\max}$ and $F2|no\text{-}wait|C_{\max}$, which can be solved optimally in polynomial time, see Sections 1.4.1 and 1.6.3. In our problem, there are two types of jobs: *regular* jobs and *no-wait* jobs. Each regular job is processed as in problem $F2|\ |C_{\max}$, i.e., it starts on machine B *no earlier than* it is completed on A. Each no-wait job is processed as in problem $F2|no\text{-}wait|C_{\max}$, i.e., it starts on machine B *immediately after* it is completed on A.

Extending standard scheduling notation, we denote the problem under consideration by $F2|reg + no\text{-}wait|C_{\max}$. Let $N_R = \{J_1, \ldots, J_{n_R}\}$ be the set of regular jobs, $|N_R| = n_R$, $N_{NW} = \{I_1, \ldots, I_{n_{NW}}\}$ be the set of no-wait jobs, $|N_{NW}| = n_{NW}$. We have $n = n_R + n_{NW}$ jobs to schedule. The processing times of the regular jobs on machines A and B are denoted by a_j and b_j, while for the no-wait jobs these times are equal to α_i and β_i, respectively.

If either the number of regular jobs or the number of the no-wait jobs is bounded by a constant q, the problem is denoted either $F2|reg + no\text{-}wait, n_R \leq q|C_{\max}$ or $F2|reg + no\text{-}wait, n_{NW} \leq q|C_{\max}$, respectively.

For each of the basic problems $F2|\ |C_{\max}$ and $F2|no\text{-}wait|C_{\max}$ there exists an optimal solution that is a permutation schedule, i.e., a schedule in which each machine processes the jobs according to the same sequence. But for problem $F2|reg+no\text{-}wait|C_{\max}$ it is not sufficient to search for an optimal schedule in the class of permutation schedules. It can be illustrated by the following example.

Example 3.1 *We are given 3 no-wait jobs I_1, I_2 and I_3 with processing*

times

$$\alpha_1 = 0, \quad \beta_1 = 3;$$
$$\alpha_2 = 2, \quad \beta_2 = 2;$$
$$\alpha_3 = 3, \quad \beta_1 = 0;$$

and one regular job J_1 *with processing times*

$$a_1 = 1, \quad b_1 = 1.$$

It is clear that in the only optimal schedule machine A processes jobs in order (I_1, J_1, I_2, I_3) *and machine B processes jobs in order* (I_1, I_2, J_1, I_3).

We restrict our consideration to the class of permutation schedules. In the case of the problem with a fixed number of no-wait jobs $F2|reg + no\text{-}wait, n_{NW} \leq q|C_{\max}$, the no-wait jobs can be treated as consecutive non-availability intervals with non-fixed running times. Hence, it can be shown that our method of constructing a dynamic programming algorithm can be extended to this problem. By fixing the starting times of the no-wait jobs in the first step of the dynamic programming algorithm, the problem under consideration becomes very similar to the flow shop problem under the resumable scenario and the presented dynamic programming algorithm requires only minor revision. A detailed discussion of this problem and methods of its solving can be found in [16].

3.7 Conclusion

In this chapter we consider the two-machine flow shop scheduling problem with availability constraints under different scenarios. The contribution of this chapter against the previously known results is summarized in Table 3.1 (for the resumable scenario) and Table 3.2 (for the semi-resumable scenario).

It can be seen that the chapter provides a fairly complete approximability classification of the relevant problems. An interesting topic for future research is whether the two-machine flow shop problem with a single hole under the semi-resumable scenario admits an FPTAS.

Structure of holes	Resumable	
	Previously known	In this chapter
$(1,0)$	DP, [89]; FPTAS, [107]	FPTAS, Section 3.3
$(0,1)$	DP, [89]; FPTAS, [107]	FPTAS, Section 3.3
$(q,0)$		DP, Section 3.2; $\rho = \frac{3}{2}$, Section 3.4
$(0,q)$	Not approximable for $q \geq 2$, [76]	DP, Section 3.2
$(1,1)$	Not approximable, [76]	

Table 3.1: Results for the two-machine flow shop sheduling problem with availability constraints under the resumable scenario

Strucure of holes	Semi-resumable	
	Previously known	In this chapter
$(1,0)$	Dynamic programming, [90] $\rho = 2$, [90]	PTAS, Section 3.5
$(0,1)$	Dynamic programming, [90] $\rho = \frac{3}{2}$, [90]	PTAS, Section 3.5
$(q,0)$	Not approximable for $q \geq 2$, [19, 88]	
$(0,q)$	Not approximable for $q \geq 2$, [19, 88]	
$(1,1)$	Not approximable, [19, 88]	

Table 3.2: Results for the two-machine flow shop sheduling problem with availability constraints under the semi-resumable scenario

We also demonstrated that the developed technique can be also applied to other flow shop problems, i.e., the flow shop problem with a fixed number of no-wait jobs.

Chapter 4

Flow Shop No-wait Scheduling

4.1 Introduction

In this chapter we study the two-machine flow shop scheduling problem with no-wait in process to minimise the makespan, provided that a machine is not available for processing during a given interval. This chapter is based on paper [81]. The results have been reported at CO'02, see [80].

For a literature review and a discussion of scheduling with machine non-availability see Chapter 2.

The remainder of this chapter is organized as follows. In Section 4.2 we show that our problem is NP-hard irrespective the scenario. Section 4.3 contains the analysis of a 3/2-approxiamtion algorithm that is applicable to any scenario. A 4/3-approximation algorithm for the resumable scenario is described and analyzed in Section 4.4. Concluding remarks are contained in Section 4.5.

4.2 Complexity and Approximability

In this section we discuss the issues of computational complexity and approximability of the two-machine flow shop no-wait problem with non-availability intervals under various scenarios.

Espinouse et al. [38, 39] prove that each of the problems $F2|no-$

$wait, h(1,0), N\text{-}Re|C_{\max}$ and $F2|no-wait, h(0,1), N\text{-}ReB2|C_{\max}$ is NP-hard. The proof essentially uses the fact that a single machine problem with a single hole to minimise the makespan is NP-hard under the non-resumable scenario, see Lee [88]. That proof technique does not allow one to establish the complexity status of the two-machine flow shop no-wait problems under other scenarios. Below we give a proof which holds for any scenario. PARTITION problem, as defined in Section 1.2.4, is used for the reduction.

Theorem 4.1 *Problem $F2|no-wait, h(1,0)|C_{\max}$ is NP-hard irrespective of the scenario of processing the crossover job.*

Proof. Given an instance of PARTITION, define the following instance of problem $F2|no-wait, h(1,0)|C_{\max}$. There are $n = r + 2$ jobs such that

$$\begin{array}{ll} a_j = 1, & b_j = e_j,\ j = 1, 2, \ldots, n; \\ a_{n+1} = 1, & b_{n+1} = E^2 - 1; \\ a_{n+2} = E^2 - 1, & b_{n+2} = E^2 + 1. \end{array}$$

Machine A is not available during the interval $[E^2 + E, 2E^2 + E]$.

We show that PARTITION has a solution if and only if for the constructed problem there exists a schedule S_0 with $C_{\max}(S_0) \leq 2E^2 + 2E + 1$.

Suppose that the subsets R_1 and R_2 form a solution to PARTITION. Then the required schedule S_0 exists and can be constructed as shown in Figure 4.1. Each machine processes the jobs in the sequence that starts with an arbitrary sequence of jobs J_j for $j \in R_1$, then the sequence of jobs (J_{n+1}, J_{n+2}), which is turn is followed by an arbitrary sequence of jobs J_j for $j \in R_2$. Each operation starts as early as possible; in particular operation $O_{n+2,B}$ is scheduled in the time interval $[E^2 + E, 2E^2 + E + 1]$.

Suppose now that schedule S_0 exists. Since total workload on machine B is equal to $2E^2 + 2E + 1$ and the smallest processing time on machine A is equal to 1 it follows that in S_0 machine B is permanently busy starting at time 1. This implies that no operation can start or complete on machine

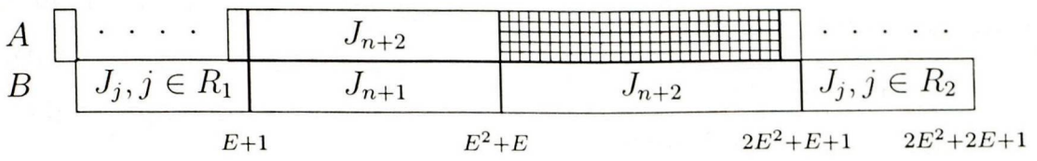

Figure 4.1: Schedule S_0

B inside the interval of non-availability of machine A. The only operation that may start no later than time $s = E^2 + E$ and complete no earlier than time $t = 2E^2 + E$ is operation $O_{n+2,B}$ since its duration exceeds the length of the hole. Notice that the operation that immediately follows the hole on machine A can be completed no earlier than time $t + 1$, i.e., the operation that follows $O_{n+2,B}$ cannot start earlier than time $t + 1$. If operation $O_{n+2,B}$ starts strictly earlier than time s and therefore completes strictly earlier than time $t + 1$, there must be idle time on B after $O_{n+2,B}$, which is impossible. Thus, in schedule S_0 operation $O_{n+2,B}$ is processed in the time interval $[E^2 + E, 2E^2 + E + 1]$. Due to the no-wait restriction, operation $O_{n+2,A}$ must be processed in the interval $[E + 1, E^2 + E]$. To avoid idle time in this interval on machine B we must schedule operation $O_{n+1,B}$ in the interval. All other operations on machine B must be processed during two time intervals $[1, E + 1]$ and $[2E^2 + E + 1, 2E^2 + 2E + 1]$. This implies that PARTITION must have a solution.

Notice that in schedule S_0 the type of scenario is irrelevant since no job is affected by the hole. ■

Theorem 4.1 can easily be modified for the case of the hole on machine B.

Notice that the problem with a single hole is solvable in $O(n \log n)$ time, provided that the hole starts at time $s = 0$. Since in this case the first job on the machine with the hole may only start after the hole, i.e., no earlier than time t, it follows that only the non-resumable scenario applies. Problem $F2|no-wait, h(1,0)|C_{\max}$ in which machine A is not available at time zero is

trivial, the corresponding problem is equivalent to problem $F2|no{-}wait|C_{\max}$ with all starting times increased by the length $\Delta = t$ of the hole.

Given an instance of problem $F2|no-wait, h(0,1)|C_{\max}$ with $s = 0$, introduce problem $F2|no-wait|C_{\max}$ with continuously available machines and an extra job J_{n+1} such that $a_{n+1} = 0$ and $b_{n+1} = \Delta$. Solve the obtained problem by the Gilmore-Gomory algorithm and find an optimal permutation $(J_k, \sigma_1, J_{n+1}, \sigma_2)$, where σ_1 and σ_2 are some sequences of jobs. The nature of the algorithm is such that an optimal permutation will start with a job with zero processing time on machine A, i.e., $a_k = 0$. If J_{n+1} is the first job, then the sequence (J_k, σ_1) is dummy and an optimal schedule for the original problem is defined by the sequence σ_2. On the other hand, if $J_k \neq J_{n+1}$, it can be seen that the sequence $(J_{n+1}, \sigma_2, J_k, \sigma_1)$ is also optimal for problem $F2|no-wait|C_{\max}$ with job J_{n+1}, so that the sequence $(\sigma_2, J_k, \sigma_1)$ specifies an optimal schedule for the original problem.

Espinouse et al. [38, 39] prove that each of the two-machine flow shop no-wait problems with at least two holes is not approximable within a constant factor, unless P=NP. Although the proof is only given for the non-resumable scenario, the same technique could be extended to any scenario and any location of the holes (either two holes on the same machines or one hole on each machine). The technique involves consideration of the instances in which in an acceptable heuristic schedule no processing must take place after the second hole, while the problem of completing all jobs before the second hole is NP-hard. Similar ideas have been used for other scheduling problems with non-availability intervals, see, e.g., [76].

From now on we restrict our attention to problems with a single hole only.

4.3 A $\frac{3}{2}$−Approximation Algorithm

In this section we consider problem $F2|no-wait, h(0,1), Sc|C_{\max}$ with an arbitrary scenario. We design an approximation algorithm that runs in $O(n^3)$

time and creates a schedule with the makespan that is at most 3/2 times the optimum.

For the non-resumable scenario $B2$, the problem has been considered by Espinouse et al. [38, 39] who have come up with several 2−approximation algorithms. Recently, Wang and Cheng [147] have given a $\frac{5}{3}-$ approximation algorithm, although they have failed to prove the tightness of their ratio bound. Our algorithm not only guarantees a better worst-case performance, but also its analysis is scenario-independent and the bound is proved tight.

The intuition behind our approach is as follows. We start with a schedule associated with a Gilmore-Gomory permutation for problem $F2|no - wait|C_{\max}$ with continuously available machines. Inserting the hole into that schedule, we identify the job that is affected by the hole. This job either starts on machine B after the hole, or (for the (semi-)resumable scenario) is interrupted by the hole. We remove that job together with a pair of other jobs from the original instance and find an optimal schedule for these three jobs with the hole. All possible pairs of jobs will be enumerated to be scheduled together with the job affected by the hole. For each choice of these three jobs their partial schedule is complemented by the Gilmore-Gomory sequence of the remaining jobs. Thus, the algorithm will create $O(n^2)$ schedules and select the best as a heuristic solution. We guarantee that among generated schedules there will be either an optimal schedule or at least one for which the partial schedule of three selected jobs is complemented by a schedule of the remaining jobs with the makespan that is at most half of the optimum.

Algorithm H1

INPUT: An instance of problem $F2|no - wait, h(0,1), Sc|C_{\max}$.

OUTPUT: A heuristic schedule S_H.

1. Temporarily disregard the non-availability interval $[s,t]$ on machine B and find schedule S_{GG} that is optimal for the resulting problem $F2|no - wait|C_{\max}$ using the algorithm of Gilmore and Gomory, see

Section 1.6 and Section 1.6.3. If necessary, renumber the jobs in such a way that in schedule S_{GG} they are processed according to the sequence $J_1, J_2, \ldots, J_n$.

2. Insert the hole $[s,t]$ on machine B into schedule S_{GG}. Identify job J_k such that operation O_{kB} is the first operation on machine B that starts no earlier than time t. Call the resulting schedule S_1.

3. If there is idle time before O_{kB}, then define $S_H = S_1$ and Stop. Otherwise, go to the next step.

4. If either the non-resumable scenario $B2$ applies or operation $O_{k-1,B}$ is not interrupted by the hole, then define $J' = J_k$; otherwise, define $J' = J_{k-1}$.

5. For each pair of jobs J_p and J_q different from J' for $1 \leq p < q \leq n$, do the following:

   (a) By enumerating all possibilities solve an auxiliary problem $F2|no-wait, h(0,1), Sc|C_{\max}$ with three jobs J', J_p and J_q and the hole $[s,t]$. Call the obtained schedule S'_{pq}.

   (b) Solve problem $F2|no-wait|C_{\max}$ for the original set of jobs with jobs J', J_p and J_q removed. Call the obtained schedule S''_{pq}.

   (c) Find schedule S_{pq} for the original problem by concatenating schedules S'_{pq} and S''_{pq}.

6. Among all found schedules output schedule S_H that has the minimum makespan and stop.

Let us estimate the running time of Algorithm H1. Step 1 requires $O(n \log n)$ time. Due to Remark 1.1, for each pair of jobs J_p and J_q in Step 5, we can derive the solution to the matching subproblem of the corresponding problem $F2|no-wait|C_{\max}$ with the reduced set of jobs by removing jobs J_p

and J_q from the solution to the matching subproblem obtained in Step 1, and this takes constant time. In order to find schedule S''_{pq} it remains to solve the patching subproblem, and that requires $O(n)$ time. Since for each J_p and J_q Step 5(a) takes constant time, total running time of Steps 5(a)-5(c) is $O(n)$, and the overall complexity of the algorithm is $O(n^3)$.

We now analyze worst-case performance of Algorithm H1. We prove that the inequality

$$\frac{C_{\max}(S_H)}{C_{\max}(S^*)} \leq \frac{3}{2} \tag{4.1}$$

holds for any instance of problem $F2|no-wait, h(0,1), Sc|C_{\max}$, where S^* is an optimal schedule for the corresponding scenario.

First, suppose that the conditions of Step 3 hold. This implies that the length of the hole does not exceed the length of the idle interval on machine B before processing job J_k in schedule S_{GG}. Thus, the insertion of the hole does not increase the makespan and for schedule S_H found in Step 3 we have that $C_{\max}(S_H) = C_{\max}(S_{GG})$, i.e., this schedule is optimal.

In the remainder of this section we assume that $C_{\max}(S_H) > C_{\max}(S_{GG})$, and the insertion of the hole into schedule S_{GG} does delay the starting times of job J_k and of all jobs that follow.

Lemma 4.1 *Let J_k be the job found in Step 2 of Algorithm H1 and in Step 4 we define $J' = J_k$. If the inequality*

$$\Delta + b_k \leq \frac{1}{2} C_{\max}(S^*)$$

holds, then (4.1) holds for $S_H = S_1$.

Proof. If the non-resumable scenario $B2$ applies, in schedule S_1 operation O_{kB} starts exactly at time t. For all other scenarios, since $s \leq R_{kB}(S_{GG})$, no portion of job J_k can be processed before the hole in schedule S_1, so that the schedule will essentially be the same as in the non-resumable case.

Thus, in the obtained schedule $R_{kB}(S_1) = t$. Denote $u = R_{k,A}(S_{GG})$ and $u' = C_{k-1,A}(S_{GG}) = C_{k-1,A}(S_1)$. The no-wait condition implies that

$$u = u' + \max\{b_{k-1} - a_k, 0\}. \tag{4.2}$$

After the insertion of the hole into schedule S_{GG} the starting times of job J_k and of all jobs $J_{k+1}, \ldots, J_n$ are delayed by $R_{k,A}(S_1) - R_{k,A}(S_{GG})$. Since we are only interested in the case that $R_{k,B}(S_1) = t$, it follows that $R_{k,A}(S_1) = t - a_k$, and the length of the delay is equal to $t - a_k - u$, so that

$$C_{\max}(S_1) = C_{\max}(S_{GG}) + t - a_k - u. \tag{4.3}$$

Since $R_{k,B}(S_1) = t$, it follows that in schedule S_1 there is no room to process job J_k on both machines before the hole, i.e., before time s. This implies that

$$u' + \max\{b_{k-1}, a_k\} + b_k > s.$$

Substituting into (4.3) yields

$$\begin{aligned} C_{\max}(S_1) &= C_{\max}(S_{GG}) + t - a_k - (\max\{b_{k-1} - a_k, 0\} + u') \\ &< C_{\max}(S_{GG}) + t - a_k - \\ &\quad (\max\{b_{k-1} - a_k, 0\} - (s - b_k - \max\{b_{k-1}, a_k\}) \\ &= C_{\max}(S_{GG}) + t - \max\{b_{k-1}, a_k\} - s + b_k + \max\{b_{k-1}, a_k\} \\ &= C_{\max}(S_{GG}) + \Delta + b_k. \end{aligned}$$

The lemma follows from the fact that $C_{\max}(S_{GG})$ is an obvious lower bound on the optimal makespan $C_{\max}(S^*)$. ■

Lemma 4.2 *Let J_k be the job found in Step 2 of Algorithm H1 and in Step 4 we define $J' = J_{k-1}$. If the inequality*

$$\Delta + b_{k-1} \leq \frac{1}{2} C_{\max}(S^*)$$

holds, then (4.1) holds for $S_H = S_1$.

Proof. Assume that the semi-resumable scenario with a parameter $\alpha_k \in [0,1]$ applies. Recall that $\alpha_k = 0$, for all $k = 1, \ldots, n$, corresponds to the resumable scenario, while the case $\alpha_k = 1$, for all $k = 1, \ldots, n$, corresponds to the non-resumable scenario $B1$.

In schedule S_1 the processing of operation $O_{k-1,B}$ is interrupted by the hole, so that the operation is processed before the hole for x_{k-1} time units and after the hole for $b_{k-1} - (1 - \alpha_{k-1})x_{k-1}$ time units.

Denote $u = R_{k,A}(S_{GG})$ and $u' = C_{k-1,A}(S_{GG}) = C_{k-1,A}(S_1)$. As in the proof of the previous lemma, (4.2) holds.

After the insertion of the hole into schedule S_{GG} the starting times of job J_k and of all jobs $J_{k+1}, \ldots, J_n$ are delayed by $R_{k,A}(S_1) - R_{k,A}(S_{GG})$. Since $C_{k,A}(S_1) = C_{k-1,B}(S_1)$, we derive

$$R_{k,A}(S_1) = u' + \Delta + b_{k-1} + \alpha_{k-1}x_{k-1} - a_k$$

so that

$$C_{\max}(S_1) = C_{\max}(S_{GG}) + u' + \Delta + b_{k-1} + \alpha_{k-1}x_{k-1} - a_k - u.$$

Substituting (4.2) yields

$$\begin{aligned} C_{\max}(S_1) &= C_{\max}(S_{GG}) + \Delta + b_{k-1} + \alpha_{k-1}x_{k-1} - a_k - \max\{b_{k-1} - a_k, 0\} \\ &< C_{\max}(S_{GG}) + \Delta + \alpha_{k-1}x_{k-1} + \min\{b_{k-1} - a_k, 0\} \\ &\leq C_{\max}(S_{GG}) + \Delta + b_{k-1}, \end{aligned}$$

which proves the lemma. ■

For job J' found in Step 4, denote its processing time on machine B by b'. We need to study the algorithm's performance provided that the inequality

$$\Delta + b' > \frac{1}{2}C_{\max}(S^*) \tag{4.4}$$

holds.

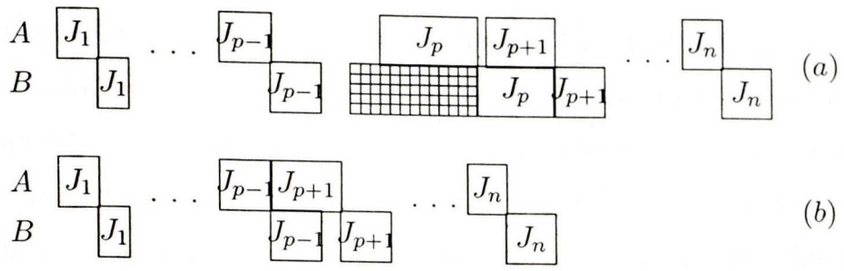

Figure 4.2: Removing the hole and job J_p from schedule S^*

Lemma 4.3 *Let J' be the job found in Step 4 of Algorithm H1. If (4.4) holds, then there exist a pair of jobs J_p and J_q such that the makespan of the flow shop no-wait schedule S''_{pq} found in Step 5(b) does not exceed $\frac{1}{2}C_{\max}(S^*)$.*

Proof. Suppose that in a certain schedule S^* that is optimal for the original problem $F2|no-wait, h(0,1), Sc|C_{\max}$ the first job that starts after the hole on machine B is denoted by J_p. Let J_q be the job that immediately follows job J' in schedule S^*. In this proof we assume that job J_q exists and job J_p is different from J'; otherwise the lemma holds for any job J_q or any job J_p, respectively.

We show that by removing the jobs J', J_p and J_q together with the hole from schedule S^* we can obtain a flow shop no-wait schedule S''_{pq} for the remaining jobs with the makespan that is at least $\Delta + b'$ time units less.

Remove the hole and job J_p from the schedule, see Figure 4.2. The starting times of all jobs that followed J_p in S^* can be decreased at least by Δ, because in S^* all these operations start later than time t and the only operation on A that is processed in the time interval $[s,t]$ is O_{pA} which is now removed. If for the (semi-)resumable scenario there exists a job that is interrupted by the hole in schedule S^*, then after the hole is removed, that job is processed with no preemption, and the length of this processing is equal to its original processing time. Call the resulting schedule S_p. Let in this schedule job J' be processed on machine B in the time interval $[\tau', \tau'']$ of length b'.

Remove the jobs J' and J_q from schedule S_p. The starting times of all jobs that followed J_q in S_p can be decreased at least by b' because in S_p all these operations start later than time τ'' and the only operation on A that is processed in the time interval $[\tau', \tau'']$ is O_{qA} which is now removed. Call the resulting schedule $\tilde{S}_{pq}$. It follows that this schedule is a feasible flow shop no-wait schedule for the remaining jobs and continuously available machines. Moreover, due to (4.4)

$$C_{\max}(\tilde{S}_{pq}) \leq C_{\max}(S^*) - \Delta - b' < \frac{1}{2} C_{\max}(S^*).$$

Schedule S''_{pq} found in Step 5(b) is an optimal schedule for the same set of jobs, and this proves the lemma. ■

Theorem 4.2 *Let S_H be a schedule found by Algorithm H1 for problem $F2|no-wait, h(0,1), Sc|C_{\max}$. Then the bound (4.1) holds, and the bound is tight.*

Proof. Owing Lemmas 4.1 and 4.2, to prove that (4.1) holds we only need to consider the case that (4.4) is valid. Taking into account Lemma 4.3, we argue as follows. Since in Step 5 we enumerate all possible pairs of jobs J_p and J_q to be removed together with job J' and the hole, it follows that at least one schedule S''_{pq} satisfies $C_{\max}(S''_{pq}) < \frac{1}{2} C_{\max}(S^*)$. In Step 5(a), for every pair of jobs J_p and J_q, we have that $C_{\max}(S'_{pq})$ is a lower bound on the makespan of an optimal schedule with the complete set of jobs and the corresponding scenario. Thus, joining schedules S'_{pq} and S''_{pq} together we obtain a feasible schedule S_H that satisfies (4.1).

To see that the bound (4.1) is tight, consider the following instance of problem $F2|no-wait, h(0,1), Sc|C_{\max}$. There are n jobs such that $a_1 = \frac{1}{2} + \frac{1}{4n}, b_1 = \frac{1}{4n}$ and $a_j = \frac{1}{2n}, b_j = \frac{1}{4n}$ for all $j = 2, \ldots, n$. The hole on machine B occupies the interval $\left[\frac{1}{2}, 1 - \frac{1}{4n}\right]$.

There exists a schedule S^* in which no job is interrupted by the hole and the jobs are processed in an arbitrary sequence with job J_1 in the last

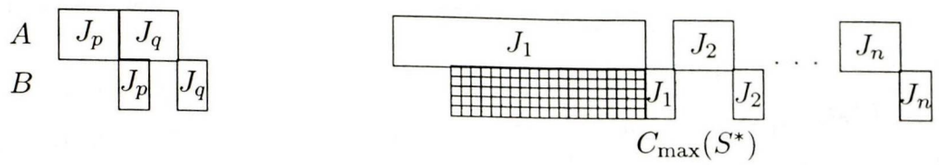

Figure 4.3: Tightness example for Algorithm H1

position. It can be verified that this schedule is optimal for any scenario and $C_{\max}(S^*) = 1$.

Any permutation defines schedule S_{GG} found in Step 1. We take the sequence $J_1, J_2, \ldots, J_n$. Inserting the hole in Step 2 we find a schedule S_1 with job J_2 as job J_k. Notice that operation O_{kB} starts at time t. Since job $J_{k-1} = J_1$ is not interrupted by the hole, in Step 4 we define $J' = J_1$.

In Step 5, if the triple of jobs to be scheduled in schedule S'_{pq} does not contain job J_1, then $C_{\max}(S'_{pq}) = t$, and provided that in the Gilmore-Gomory sequence for the remaining jobs job J_1 occupies the last position, we have that $C_{\max}(S_{pq}) = t + a(N) - 4\left(\frac{1}{2n}\right) + \frac{1}{4n} = 2 - \frac{9}{4n}$. Otherwise, if the triple of job in schedule S'_{pq} contains job J_1, then J_1 is processed last, so that $C_{\max}(S'_{pq}) = t + b_1 = 1$. Concatenating an arbitrary sequence of the remaining jobs we obtain that $C_{\max}(S_{pq}) \leq 1 + (n-3)\frac{1}{2n} + \frac{1}{4n} = \frac{3}{2} - \frac{5}{4n}$; see Figure 4.3.

Thus, as n grows to approach infinity the ratio $C_{\max}(S_H)/C_{\max}(S^*)$ goes to 3/2.

Notice that the example above demonstrates that we cannot improve the worst-case performance of Algorithm H1 by arranging complete enumeration of more than two jobs along with the chosen job J'. As seen from Figure 4.3, more jobs can be processed in the gap before job J_1, however, this will not dramatically reduce the makespan. ■

It is easy to convert Algorithm H1 for the case of the hole on machine A. In the description of the algorithm the only change concerns the choice of job J_k : now it is the job that starts on A after the hole. Besides, in Step 4 the non-resumable scenario $B2$ is not applicable.

The converted version of Algorithm H1 is a $\frac{3}{2}-$approximation algorithm for problem $F2|no-wait, h(1,0), Sc|C_{\max}$ which can be proved by the statements similar to Lemmas 4.1-4.3. In the formulation of the analogue of Lemma 4.1 we should change b_k for a_k, while in the analogue of Lemma 4.2 we change b_{k-1} for a_{k-1}. The proofs of these analogues are quite similar to the original Lemmas. In the formulation of the analogue of Lemma 4.3, we should replace b' by a', where a' is the processing time of job J' on machine A. In the proof of this statement, job J_p is chosen to be the job that immediately precedes the hole on A in an optimal schedule S^*, while S_q is the job that immediately precedes job J' in S^*.

4.4 Heuristic for the Resumable Scenario

In this section we consider problem $F2|no-wait, h(0,1), Re|C_{\max}$ with the resumable scenario. We design a $\frac{4}{3}-$approximation algorithm that requires $O(n^3)$ time.

A number of approximation algorithms for the two-machine shop problems with a single hole under the resumable scenario are known. The best of these heuristics are $\frac{4}{3}-$approximation algorithms, see [32, 89] for the flow shop without the no-wait restriction and [18] for the open shop.

Our algorithm in many aspects is quite similar to Algorithm H1. The points of difference include a special treatment of a job with large total processing time, as well as a different arrangement for enumeration of the jobs to be scheduled separately with the hole.

Algorithm H2

INPUT: An instance of problem $F2|no-wait, h(0,1), Re|C_{\max}$.

OUTPUT: A heuristic schedule S_R.

1. Temporarily disregard the non-availability interval $[s,t]$ on machine B and using algorithm of Gilmore and Gomory find schedule S_{GG} that is

optimal for problem $F2|no-wait|C_{\max}$ with the original set of jobs N, see Section 1.6 and Section 1.6.3 for details. Insert the hole $[s,t]$ on machine B into schedule S_{GG} possibly interrupting one of the operations on B. Call the resulting schedule S_0.

2. Find job J_r such that

$$a_r + b_r = \max\{a_j + b_j | J_j \in N\}.$$

For each choice of p and q, by enumerating all possibilities solve an auxiliary problem $F2|no-wait, h(0,1), Re|C_{\max}$ with the jobs J_p, J_q and J_r and the hole $[s,t]$. Concatenate the obtained partial schedule with the schedule that is optimal for problem $F2|no-wait|C_{\max}$ for the original set N of jobs with these three jobs and the hole removed. Call the best of the obtained schedules $\bar{S}$.

3. For each job $J_k \in N$ do the following:

   (a) Find schedule S^k_{GG} that is optimal for problem $F2|no-wait|C_{\max}$ with continuously available machines and the set of jobs $N\backslash\{J_k\}$.

   (b) Insert the hole $[s,t]$ on machine B into schedule S^k_{GG}. Insert job J_k in such a way that operation O_{kB} is the first operation on machine B that starts no earlier than time t. The processing of the operation that precedes O_{kB} can be interrupted by the hole. Call the resulting schedule S_k. If necessary, renumber the jobs in such a way that in S_k job J_k immediately follows job J_{k-1} and is immediately followed by job J_{k+1}.

   (c) Compute

$$W_k = \max\{b_k + \Delta, b_{k-1} + \Delta, a_{k+1} + \Delta\}.$$

   (d) Depending on the value of W_k, select the objects shown in the table below. For each selection, i.e., for each choice of

p, by enumerating all possibilities solve an auxiliary problem $F2|no-wait, h(0,1), Re|C_{\max}$ with the selected jobs and the hole $[s,t]$. Concatenate the obtained partial schedule with the schedule that is optimal for problem $F2|no-wait|C_{\max}$ for the original set N of jobs with the selected objects removed. Call the best of the obtained schedules S'_k.

W_k	Objects to be selected
$b_k + \Delta$	the hole, J_k, J_p for $p \neq k$
$b_{k-1} + \Delta$	the hole, J_{k-1}, J_k, J_p for all $p \neq k$
$a_{k+1} + \Delta$	the hole, J_k, J_{k+1}, J_p for all $p \neq k$

4. Among all found schedules output schedule S_R that has the minimum makespan and stop.

Similarly to Algorithm H1, the running time of Algorithm H2 is $O(n^3)$. Step **2** generates $O(n^2)$ schedules, in Step 3 $O(n)$ schedules are generated for each k. Each of these $O(n^2)$ schedules requires finding a Gilmore-Gomory sequence, which takes $O(n)$ time per schedule (as before, only the patching is required, while the matching can be performed once and for all schedules). Thus, the overall time complexity of the algorithm is $O(n^3)$.

We now analyze worst-case performance of Algorithm H2. We prove that the inequality

$$\frac{C_{\max}(S_R)}{C_{\max}(S^*)} \leq \frac{4}{3} \tag{4.5}$$

holds for instance of problem $F2|no-wait, h(0,1), Re|C_{\max}$, where S^* is an optimal schedule for the resumable scenario.

In Step 1 of the algorithm, if the insertion of the hole into schedule S_{GG} interrupts the processing of one of the operations, this delays the completion time of the succeeding jobs by at most Δ, so that

$$C_{\max}(S_0) \leq C_{\max}(S_{GG}) + \Delta.$$

In the remainder of this section we assume that

$$\Delta > \frac{1}{3} C_{\max}(S^*), \tag{4.6}$$

since otherwise (4.5) holds for $S_R = S_0$.

We start with schedule $\bar{S}$ found in Step 2 of the algorithm.

Lemma 4.4 *If the condition*

$$a_r + b_r > \frac{2}{3} C_{\max}(S^*) \tag{4.7}$$

holds, then for $S_R = \bar{S}$ the bound (4.5) is valid.

Proof. First, we show that if (4.7) holds then job J_r is unique. Suppose that there exists another job J_x with total processing times that exceeds $\frac{2}{3} C_{\max}(S^*)$. In any optimal schedule, neither job J_r nor J_x can be completed before the hole or interrupted by the hole due to (4.6). Because of the no-wait condition, the first of these jobs cannot be completed on machine A earlier than time t. Thus, the second job must be totally processed starting at time t or later. This, however, is also impossible since $t > \frac{1}{3} C_{\max}(S^*)$ due to (4.6).

Consider an arbitrary optimal schedule S^*. As proved above, job J_r starts on B after the hole. Moreover, it can be seen that J_r is the first job that starts after the hole, since otherwise it must start on machine A later than time t. Let J_p be the job that precedes the hole and let J_q be the job that immediately follows job J_r in schedule S^*. In this proof we consider the general case that both jobs J_p and J_q exist; otherwise the proof can be suitable modified.

Similar to the proof of Lemma 4.3, it can be shown that by removing the jobs J_r, J_p and J_q together with the hole from schedule S^* and ordering the remaining jobs according to the Gilmore-Gomory sequence we obtain a flow shop no-wait schedule $\tilde{S}_{pq}$ for the remaining jobs with the makespan that is at least $a_r + b_r$ time units less, i.e., due to (4.7)

$$C_{\max}(\tilde{S}_{pq}) \leq C_{\max}(S^*) - a_r - b_r < \frac{1}{3} C_{\max}(S^*).$$

Since in Step 2 we enumerate all possible pairs of jobs J_p and J_q to be removed together with job J_r and the hole, it follows that at least one

$\max\{a_{k+1}, b_k\}$	$\max\{a_{k+1}, b_{k-1}\}$	$\max\{a_k, b_{k-1} + \Delta\}$	LHS of (4.8)
a_{k+1}	a_{k+1}	a_k	a_k
a_{k+1}	a_{k+1}	$b_{k-1} + \Delta$	$b_{k-1} + \Delta$
a_{k+1}	b_{k-1}	a_k	a_k
a_{k+1}	b_{k-1}	$b_{k-1} + \Delta$	$a_{k+1} + \Delta$
b_k	a_{k+1}	a_k	$a_k + b_k$
b_k	a_{k+1}	$b_{k-1} + \Delta$	$b_k + \Delta$
b_k	b_{k-1}	a_k	$a_k + b_k$
b_k	b_{k-1}	$b_{k-1} + \Delta$	$b_k + \Delta$

Table 4.1: The proof of Lemma 4.5

schedule $\tilde{S}_{pq}$ satisfies $C_{\max}(\tilde{S}_{pq}) < \frac{1}{3}C_{\max}(S^*)$. Furthermore, the makespan of an optimal schedule for the jobs J_r, J_p and J_q is a lower bound on the makespan of an optimal schedule with the complete set of jobs. By appending schedule $\tilde{S}_{pq}$ we obtain a required schedule. ■

Let S^* be an optimal schedule for problem $F2|no-wait, h(0,1), Re|C_{\max}$ in which job J_k is the first job that starts on machine B after the hole. The following lemmas deal with schedule S_k found in Step 3 of Algorithm H2. Recall that in S_k the jobs are numbered in such a way that job J_k immediately follows job J_{k-1} and is immediately followed by job J_{k+1}.

Lemma 4.5 *For schedule S_k, the inequality*

$$\max\{a_{k+1}, b_k\} - \max\{a_{k+1}, b_{k-1}\} + \max\{a_k, b_{k-1} + \Delta\} > \frac{2}{3}C_{\max}(S^*) \quad (4.8)$$

implies that either

$$a_k + b_k > \frac{2}{3}C_{\max}(S^*)$$

or

$$W_k > \frac{2}{3}C_{\max}(S^*), \quad (4.9)$$

where W_k is computed in Step 3(c) of Algorithm H2.

Proof. See Table 4.1 for the left-hand sides of (4.8), which are at least as large as either $a_k + b_k$ or W_k. In several cases the values in the last column of the table are obtained by disregarding the negative terms. ■

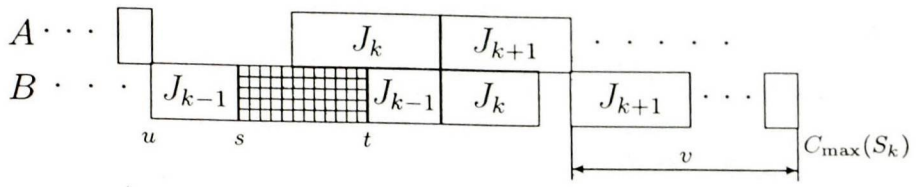

Figure 4.4: Schedule S_k

Lemma 4.6 *Suppose that for schedule S_k the condition (4.9) does not hold. Then the bound (4.5) is valid for $S_R = S_k$.*

Proof. In schedule S^k_{GG} job J_{k+1} immediately follows job J_{k-1}. Denote $u = C_{k-1,A}(S^k_{GG})$. It follows that

$$C_{\max}(S^k_{GG}) = u + \max\{a_{k+1}, b_{k-1}\} + v,$$

where v is the 'tail' of the schedule that includes the processing of operation $O_{k+1,B}$ and the jobs that follow job J_{k+1}.

Assume that the lemma does not hold, i.e., $C_{\max}(S_k) > \frac{4}{3}C_{\max}(S^*)$.

First, suppose that the makespan of the schedule obtained after the insertion of the hole remains equal to $C_{\max}(S^k_{GG})$. Thus, operation $O_{k+1,A}$ still starts at time u and is completed after time t, i.e., $a_{k+1} \geq \Delta$. However, since $W_k \geq a_{k+1} + \Delta$ the conditions of the lemma do not hold due to (4.6).

Thus, in schedule S_k operation $O_{k-1,B}$ is interrupted by the hole and operation $O_{k,A}$ is completed after time t, in fact $C_{kA}(S_k) = u + \max\{b_{k-1} + \Delta, a_k\}$, see Figure 4.4. Further, $C_{k+1,A}(S_k) = C_{kA}(S_k) + \max\{a_{k+1}, b_k\}$ and we can write

$$\begin{aligned} C_{\max}(S_k) &= u + \max\{b_{k-1} + \Delta, a_k\} + \max\{a_{k+1}, b_k\} + v \\ &= C_{\max}(S^k_{GG}) + \max\{b_{k-1} + \Delta, a_k\} + \\ &\quad \max\{a_{k+1}, b_k\} - \max\{a_{k+1}, b_{k-1}\}. \end{aligned}$$

Recall that there exists an optimal schedule S^* in which job J_k is processed on B immediately after the hole. Remove that job and the hole

from schedule S^* and reduce the starting times of the jobs that follow J_k appropriately. According to (4.6), the makespan of the obtained schedule is no larger than $\frac{2}{3}C_{\max}(S^*)$. Since schedule S^k_{GG} is an optimal no-wait schedule for the same set of jobs, we derive that

$$C_{\max}(S^k_{GG}) < \frac{2}{3}C_{\max}(S^*).$$

Thus, inequality (4.8) must hold to guarantee that $C_{\max}(S_k) > \frac{4}{3}C_{\max}(S^*)$. According to Lemma 4.5, this contradicts the conditions of the lemma under consideration. ■

Owing Lemmas 4.4 and 4.6, we only need to consider the case that (4.9) holds. In this case a required schedule can be found by running Step 3 of the algorithm. The step distinguishes between three possibilities, depending on the value of W_k. In each of these situations the actions are similar and also resemble those in Step 2 of Algorithm H2 and Step 5 of Algorithm H1. The proof of the correctness of the algorithm is quite similar to those of Lemmas 4.3 and 4.4.

Lemma 4.7 *If for schedule S_k the inequality (4.9) is valid, then (4.5) holds for $S_R = S'_k$.*

Proof. Recall that in an optimal schedule S^* the first job that starts after the hole on machine B is J_k.

If $W_k = b_k + \Delta$, then let J_p be the job that immediately follows job J_k in schedule S^*, so that operations $O_{k,B}$ and $O_{p,A}$ overlap.

If $W_k = b_{k-1} + \Delta$, then let J_p be the job that immediately follows job J_{k-1} in schedule S^*, so that operations $O_{k-1,B}$ and $O_{p,A}$ overlap.

If $W_k = a_{k+1} + \Delta$, then let J_p be the job that immediately precedes job J_{k+1} in schedule S^*, so that operations $O_{p,B}$ and $O_{k+1,A}$ overlap.

Similar to the proof of Lemmas 4.3 and 4.4, it can be shown that by removing the jobs selected in Step 3(d) for the corresponding W_k together with the hole from schedule S^* and ordering the remaining jobs according

to the Gilmore-Gomory sequence we obtain a flow shop no-wait schedule $\tilde{S}$ with $C_{\max}(\tilde{S}) < \frac{1}{3}C_{\max}(S^*)$. The fact that such a schedule will be found is guaranteed by full enumeration of jobs J_p. In any case schedule $\tilde{S}$ is appended to a schedule with the makespan that serves as a lower bound on the makespan of an optimal schedule with the complete set of jobs. ■

Theorem 4.3 *Let S_R be a schedule found by Algorithm H2 for problem $F2|no-wait, h(0,1), Re|C_{\max}$. Then the bound (4.5) holds, and the bound is tight.*

Proof. Lemma 4.4 addresses the case of a job with large total processing time. If such a job does not exists, then the algorithm is analyzed in Lemmas 4.5-4.7 proved under the assumption that job J_k is the job that follows the hole in some optimal schedule. We have demonstrated that either schedule S_k or S'_k delivers a heuristic solution within the required bound of 4/3. Since Step 3 of the algorithm is organized as a loop with respect to k, it follows that these schedules will be found.

To see that the bound (4.5) is tight, consider the following instance of problem $F2|no-wait, h(0,1), Re|C_{\max}$. There are n jobs such that

$$\begin{array}{ll} a_1 = \frac{1}{3n}, & b_1 = \frac{1}{3} + \frac{1}{3n}; \\ a_2 = \frac{2}{3} + \frac{1}{6n}, & b_2 = \frac{1}{6n}; \\ a_j = \frac{1}{3n}, & b_j = \frac{1}{6n},\ j = 3, \ldots, n. \end{array}$$

The hole on machine B occupies the interval $\left[\frac{2}{3} - \frac{1}{6n}, 1 - \frac{1}{3n}\right]$.

There exists a schedule S^* in which jobs J_1 and J_2 are processed in this order and occupy the last two positions. Operation $O_{1,B}$ starts at time $\frac{1}{3} - \frac{1}{3n}$, and then is interrupted by the hole and resumed at time t. It can be verified that this schedule is optimal and $C_{\max}(S^*) = 1$.

Any permutation in which jobs J_2 immediately follows job J_1 defines schedule S_{GG} found in Step 1. We take the sequence $J_1, J_2, \ldots, J_n$. By inserting the hole in Step 1 we find a schedule S_0. In this schedule job J_2

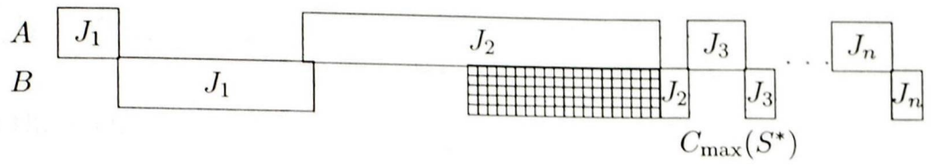

Figure 4.5: Tightness example for Algorithm H2

start on B at time t and completes at $1-\frac{1}{6n}$. When the other jobs are added, we obtain $C_{\max}(S_0) = t + (n-2)\frac{1}{3n} + \frac{1}{6n} = \frac{4}{3} - \frac{5}{6n}$.

In Step 2, job J_2 will be chosen as job J_r. Any triple of jobs containing job J_2 cannot be completed earlier than time $t + \frac{1}{6n} = 1 - \frac{1}{6n}$. For schedule $\bar{S}$, we obtain $C_{\max}(\bar{S}) \geq 1 - \frac{1}{6n} + (n-3)\frac{1}{3n} + \frac{1}{6n} = \frac{4}{3} - \frac{1}{n}$.

Since J_2 is the job that is processed on B after the hole in any optimal schedule, the best schedule found in Step 3 will be as good as the better of the schedules S_2 and S_2'.

In schedule S_{GG}^2 in Step 3(a) job J_1 cannot be in the last position, so that $C_{\max}(S_{GG}^2) = (n-2)\frac{1}{3n} + \frac{1}{3} + \frac{1}{3n} + \frac{1}{6n} = \frac{2}{3} - \frac{1}{6n} = s$. Thus, in schedule S_2 job J_2 starts on A at time $s - \frac{1}{6n}$ and completes on B at time $\frac{4}{3}$.

Since in S_2 job $J_k = J_2$, job J_{k+1} does not exist and J_{k-1} is one of the jobs $J_3, \ldots, J_n$, we may assume that $W_k = b_k + \Delta$. In the best schedule found by full enumeration with respect to p, job J_2 will be removed along with job $J_p = J_1$, and these two jobs together with the hole can be completed by time $1 - \frac{1}{6n}$. Schedule S_2' is obtained by appending an arbitrary sequence of the remaining jobs, so that $C_{\max}(S_2') = 1 - \frac{1}{3n} + (n-2)\frac{1}{3n} + \frac{1}{6n} = \frac{4}{3} - \frac{5}{6n}$, see Figure 4.5.

Thus, as n grows to approach infinity the ratio $C_{\max}(S_R)/C_{\max}(S^*)$ goes to 4/3. ■

It is easy to convert Algorithm H2 for the case of the hole on machine A. The converted version of Algorithm H2 is a $\frac{4}{3}$–approximation algorithm for problem $F2|no-wait, h(1,0), Re|C_{\max}$.

4.5 Conclusion

In this chapter we have studied the two-machine flow shop no-wait scheduling problem to minimise the makespan, provided that a machine is not available during a given time interval (the hole). We have considered all possible scenarios of handling the job affected by the hole. One of our algorithms is applicable to all scenarios and delivers a schedule with the makespan that is at most 3/2 times the optimal value. For the resumable scenario we have presented a 4/3-approximation algorithm.

It remains to find out whether the problem admits a (fully-)polynomial approximation scheme for scenarios other than N-Res B2.

Chapter 5

Open Shop Scheduling

5.1 Introduction

In this chapter, we concentrate on the open shop scheduling model which is one of the classical models for multi-stage processing systems. This chapter is based on paper [78], these results have been reported at MAPSP'03, see [79]. A review of the literature is given in Chapter 2.

Section 5.2 contains the problem formulation and discusses various preliminary matters. In Section 5.3 we present a PTAS for the open shop problem with several holes on one of the two machines. Section 5.4 describes a PTAS for the two-machine open shop with a single hole on each machine. We conclude with a short summary in Section 5.5.

5.2 Preliminaries

We denote the problem of minimising the makespan in the resumable two-machine open shop by $O2|h(q_A, q_B), Re|C_{\max}$, provided that there are q_A holes on machine A and q_B holes on machine B.

Recall that problem $O2|h(q_A, q_B), Re|C_{\max}$ is NP-hard in the ordinary sense for $q_A + q_B \geq 1$, and is not approximable within a finite factor for $q_A \geq 1$, $q_B \geq 1$ and $q_A + q_B \geq 3$; see [17].

For any schedule S^* that is optimal for problem $O2|h(q_A, q_B), Re|C_{\max}$,

where q_A denotes the number of holes on machine A and q_B denotes the number of hoes on machine B, the following lower bound

$$C_{\max}(S^*) \geq \max\{a(N), b(N)\} \tag{5.1}$$

holds.

For terminological convenience, let us agree that if a machine is said to be idle in a certain time interval, this implies that it is available in this interval and does not process a job; in other words, non-availability periods are not included into idle periods on a machine.

In many cases, the lower bound (5.1) can be refined. For example, if in a schedule some machine is not idle with possible interruption of some operations by the holes then the completion time of any job on that machine is a lower bound on the optimal makespan.

For problem $O2|h(q_A, q_B), Re|C_{\max}$ it is sufficient to restrict the search for an optimal schedule to the class $\mathcal{S}(N)$ of the schedules of the following structure:

- set N of jobs is partitioned into two subsets N_{AB} and N_{BA}, one of which may be empty, where the jobs of set N_{AB} are assigned the processing route (A, B) and the jobs of set N_{BA} are assigned the route (B, A);
- on each machine the jobs of each set N_{AB} and N_{BA} are processed as a block without intermediate idle time with a possible interruption of a job by a hole;
- the jobs of set N_{AB} on both machines follow the same sequence $\varphi(N_{AB})$ start on machine A at time zero and on machine B as early as possible;
- the jobs of set N_{BA} on both machines follow the same sequence $\psi(N_{BA})$ start on machine B at time zero and on machine A as early as possible.

If an optimal schedule is associated with a partition $N_{AB} \cup N_{BA}$ does not belong to this class, then using standard interchange argument we can also

the order of jobs to achieve the separation of the sets N_{AB} and N_{BA} into individual blocks on each machine, and then permute the jobs within each block N_{AB} or N_{BA} to achieve the same sequence on each machine.

Lemma 5.1 *For problem* $O2|h(q_A, q_B), Re|C_{\max}$ *there exists an optimal schedule such that machine* A *processes the sequence of jobs* $(\varphi(N_{AB}), \psi(N_{BA}))$ *and machine* B *processes the sequence of jobs* $(\psi(N_{BA}), \varphi(N_{AB}))$.

Proof. Consider an optimal schedule S^* in which machine A starts processing with 3 blocks:

1. Block B_1 of jobs which follow route (A, B);
2. Block B_2 of jobs which follow route (B, A);
3. Block B_3 of jobs which follow route (A, B);

and the cardinality of B_1 is maximal. If we interchange blocks B_2 and B_3 on machine A then the starting times of the jobs from block B_2 will not decrease and the completion times of the jobs from block B_3 will not increase. Hence this schedule will obey the flow shop condition as well as the original schedule and this interchange on machine A does not create any clashes with machine B. Moreover, since we consider the resumable scenario the total completion time of these blocks on machine A remains the same as in the original optimal schedule and since this interchange does not affect the other machine we deduce that the makespan of this schedule is equal to the makespan of the original optimal schedule. Hence, we receive another optimal schedule and this fact contradicts the selection of an optimal schedule S^*. We can conclude that block B_3 is empty.

Analogously we can prove that a similar result holds for machine B. We derive that there exists an optimal schedule in which machine A first processes jobs of set N_{AB} and then jobs of set N_{BA} while machine B first

processes jobs of set N_{BA} and then jobs of set N_{AB}. Notice that scheduling the jobs of set N_{AB} (and of set N_{BA}) alone reduces to the corresponding flow shop problem, for which it is known that the sequence of jobs is the same on both machines, see Lee [89]. This fact proves the Lemma. ■

A similar, but more elaborate schedule structure is used by Lorigeon et al. [100] in their dynamic programming algorithm for problem $O2|h(q_A, q_B), Re|C_{\max}$ with $q_A + q_B = 1$. They prove an important statements about the structure of an optimal schedule. An optimal schedule can be described as follows. The set of jobs N is split into four disjoint subsets XP, YP, XS and YS, and for these subsets the following propositions are true.

Proposition 5.1 *(by Lorigeon et al. [100]) There exists an optimal solution such that on machine A the sequence is defined by (XP, YP, XS, YS) and on machine B the sequence is defined by (YP, YS, XP, XS) and the order of jobs in XP, YP, XS and YS is the same on both machines.*

Proposition 5.2 *(by Lorigeon et al. [100]) At least one of the two subsets YP and XS is empty.*

In a schedule from class $\mathcal{S}(N)$ either each machine is not idle starting at time zero or there may be a single idle period on one of the machines, since some job of sequence $\varphi(N_{AB})$ may start on B exactly when completed on A (this causes the idle period on B) or some job of sequence $\psi(N_{BA})$ may start on A exactly when completed on B (this induces the idle period on A).

Our polynomial-time approximation schemes for problems $O2|h(1,1), Re|C_{\max}$ and $O2|h(0, q_B), Re|C_{\max}$ have the same common feature. Define $z = \lceil \frac{1}{\varepsilon} \rceil$, select z jobs with largest processing times on machine A and z jobs with largest processing times on machine B. Denote the set of all selected jobs by Z and call the jobs of this set *big*. By full enumeration we find a schedule S_z that is optimal for the corresponding problem

$O2|h(1,1), Re|C_{\max}$ or $O2|h(0,q_B), Re|C_{\max}$ with the same distribution of holes as in the original instance and the set of jobs Z. Schedule S_z will be sought for in class $\mathcal{S}(Z)$. Our approximation schemes use the sequences $\varphi(Z_{AB})$ and $\psi(Z_{BA})$ associated with S_z to construct an approximate solution to the problem with the original set of jobs.

Define $Y = N \backslash Z$ and call the jobs of this set *small*. For each job $J_j \in Y$ we have that $a_j \leq \frac{1}{z}a(N)$ and $b_j \leq \frac{1}{z}b(N)$. For a given $\varepsilon > 0$, the choice of z implies that

$$\max\{a_j, b_j\} \leq \frac{1}{z}C_{\max}(S^*) \leq \varepsilon C_{\max}(S^*) \tag{5.2}$$

for each job $J_j \in Y$.

5.3 Several Holes on One Machine

In this section we present and analyze a PTAS for problem $O2|h(0,q_B), Re|C_{\max}$. Our algorithm starts with splitting the set of jobs N into two subsets of big and small jobs and finding an optimal schedule for the big jobs. The length of that schedule produces a lower bound on the optimal makespan. The purpose of the subsequent steps is to create a schedule in which the completion time of any job on machine B does not exceed the optimal makespan. The completion time of machine A never exceeds the optimal makespan by more than the length of a small operation.

In the description of our algorithms the phrase "a sequence of jobs is processed on machine L starting at time τ" means that the jobs of that sequence are processed without intermediate idle time with possible interruptions by the holes.

Algorithm H0KB

INPUT: Problem $O2|h(0,q_B), Re|C_{\max}$ and an $\varepsilon > 0$

OUTPUT: A heuristic schedule S_H

1. Define $z = \lceil \frac{1}{\varepsilon} \rceil$ and determine the sets Z and Y of big and small jobs.

2. By full enumeration find a schedule S_z from the class $\mathcal{S}(Z)$ that is optimal for problem $O2|h(0, q_B), Re|C_{\max}$ with Z as the set of jobs. Assume that this schedule is characterized by the partition $Z_{AB} \cup Z_{BA}$, and denote the corresponding sequences of jobs by $\varphi(Z_{AB})$ and $\psi(Z_{BA})$.

3. Construct schedule S_H as follows:

   (a) If necessary, renumber the jobs of set N so that $Y = \{J_1, J_2, \ldots, J_y\}$, where $y = |Y|$. On machine B process the sequence $(\psi(Z_{BA}), (J_1, J_2, \ldots, J_y))$ of jobs starting at time zero. On machine A start the sequence $(J_y, J_{y-1}, \ldots, J_{k+1})$ of jobs at time zero, where either $k = 0$, if scheduling the block of jobs $(J_y, J_{y-1}, \ldots, J_1)$ on A produces no clashes, i.e.,

   $$C_{i,A} \leq R_{i,B}$$

   for all $i = 1, 2, \ldots, y$, or $k \geq 1$ is the largest integer such that job k cannot be started on machine A at time $C_{k+1,A}$ due to the clash with the processing of that job on machine B. Call the obtained partial schedule S'. Denote $\tau' = C_{k+1,A}(S')$ and $\tau'' = C_{yB}(S')$. For time t, $t < \tau''$, let $\delta(t)$ denote the total length of the availability periods of machine A during the interval $[t, \tau'']$.

   (b) If all jobs of sequence $\varphi(Z_{AB})$ can be completed on machine A in the interval $[\tau', \tau'']$, i.e., $a(Z_{AB}) \leq \delta(\tau')$, then on machine A process the sequence of jobs $\varphi(Z_{AB})$ starting at time τ', followed by the sequence $(\psi(Z_{BA}), (J_{k-1}, \ldots, J_1))$ which starts as early as possible. Complete schedule S_H by assigning job k to start on A as early as possible and the sequence $\varphi(Z_{AB})$ to start on B at time τ''.

(c) If $a_y + a(Z_{AB}) > \delta(0)$ then start the sequence $\varphi(Z_{AB})$ on A at time zero followed by any sequence of jobs of set $Z_{BA} \cup \{J_1, \ldots, J_y\}$ that starts at time $\max\{a(Z_{AB}), \tau''\}$. Complete schedule S_H by starting the sequence $\varphi(Z_{AB})$ on machine B as early as possible after time τ''.

(d) Otherwise, i.e., if $\delta(0) \geq a(Z_{AB}) + a_y > \delta(\tau')$, determine a job J_l, $k+1 \leq l \leq y$, such that

$$\sum_{j=l+1}^{y} a_j + a(Z_{AB}) \leq \delta(0), \sum_{j=l}^{y} a_j + a(Z_{AB}) > \delta(0). \tag{5.3}$$

On machine A, process the sequence $(J_y, J_{y-1}, \ldots, J_{l+1})$ of jobs starting at time zero. Complete schedule S_H by assigning the sequence $(\varphi(Z_{AB}), \psi(Z_{BA}), (J_l, J_{l-1}, \ldots, J_1))$ in such a way that the last job in sequence $\varphi(Z_{AB})$ completes on A at time τ'', and start the sequence $\varphi(Z_{AB})$ on machine B at time τ''.

4. Output schedule S_H and stop.

Notice that although machine A is continuously available in the problem under consideration, in the description of Step 3 of the algorithm we define $\delta(t)$ in terms of the total length of the availability periods of machine A. This is done in order to be able to use Step 3 of Algorithm H0KB as a part of the PTAS for problem $O2|h(1,1)|C_{\max}$ in the following section.

Theorem 5.1 *For problem* $O2|h(0, q_B), Re|C_{\max}$, *Algorithm H0KB is a polynomial approximation scheme.*

Proof. We prove that

$$\frac{C_{\max}(S_H)}{C_{\max}(S^*)} \leq 1 + \frac{1}{z}. \tag{5.4}$$

Suppose that the condition of Step 3(b) holds. All jobs of set Z_{AB} are completed on A earlier than any of them starts on B. In this case the total idle time on machine B is not longer than in any optimal schedule.

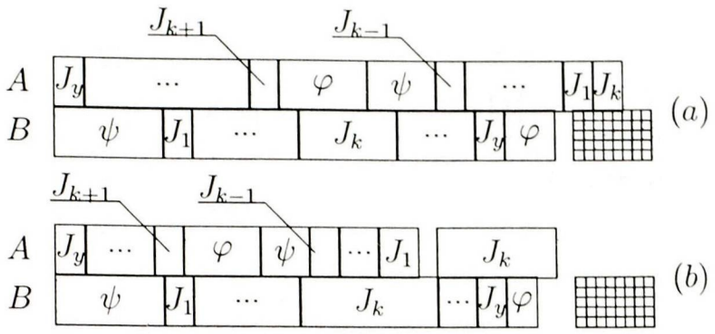

Figure 5.1: Schedule S_H found in Step 3(b) for $k > 0$.

If $k = 0$, then in schedule S_H either there is no idle time on A or machine A is idle before processing a job of set Z_{BA}. In the latter case, all jobs of set Y and all jobs of set Z_{AB} are completed on A earlier than the jobs of set Z_{BA} complete on B. This implies that $C_A(S_H) = C_A(S_z) \leq C_{\max}(S^*)$. Thus, if $k = 0$ schedule S_H is optimal.

Assume that $k > 0$, i.e., there is no idle time on A until all jobs other than job J_k are completed. Determine the earliest starting time τ of job J_k on machine A.

If $C_{1A}(S_H) \geq C_{kB}(S_H)$, then job J_k can start on A at time $\tau = C_{1A}(S_H)$. This produces no clashes, machine A has no idle time and S_H is optimal; see Figure 5.1(a). Otherwise, define $\tau = C_{kB}(S_H)$, so that $C_{\max}(S_H) \leq C_{kB}(S_H) + a_k \leq C_{\max}(S^*) + a_k$. Since $J_k \in Y$, due to (5.2) we obtain (5.4); see Figure 5.1(b). Notice that this figure and the following figure do not show the non-availability intervals on machine B, except one drawn as a shaded rectangle.

Suppose now that the condition of Step 3(c) holds. In schedule S_H, if the jobs of set Z_{AB} start on machine B at time τ'', then there is no idle time on B; otherwise, $C_B(S_H) = C_B(S_z) \leq C_{\max}(S^*)$. On machine A, either there is no idle time and the schedule is optimal or the sequence of jobs of set $Z_{BA} \cup \{J_1, \ldots, J_y\}$ starts at time τ''. In the latter case, total idle time on A

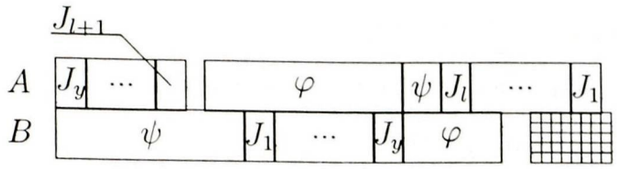

Figure 5.2: Schedule S_H found in Step 3(d)

does not exceed a_y, i.e., $C_{\max}(S_H) \leq a(N) + a_y \leq C_{\max}(S^*) + a_y$ and (5.4) follows from (5.1) and (5.2) for $J_j = J_y$.

Finally, assume that the condition of Step 3(d) holds. In schedule S_H machine B is not idle, while machine A is permanently busy starting at time τ'', and the idle time on A in the interval $[0, \tau'']$ does not exceed a_l due to (5.3). Thus, $C_A(S_H) \leq C_{\max}(S^*) + a_l$ and (5.4) holds; see Figure 5.2.

The number of jobs of set Z does not exceed $2\left\lceil \frac{1}{\varepsilon} \right\rceil$. Thus, finding a schedule S_z in Step 2 by full enumeration requires constant time for a fixed ε. The other steps of the algorithm require $O(n)$ time. We conclude that Algorithm H0KB is a polynomial-time approximation scheme. ■

It is evident that Algorithm H0KB can be converted to a PTAS for problem $O2|h(q_A, 0), Re|C_{\max}$.

5.4 One Hole on Each Machine

In this section we present and analyze a PTAS for problem $O2|h(1,1), Re|C_{\max}$ with a single hole on each machine. Let $[s_A, t_A]$ and $[s_B, t_B]$ be the holes on machine A and machine B, respectively. Without loss of generality, assume that $t_B \geq t_A$; otherwise, the machines can be appropriately renamed.

The PTAS for problem $O2|h(1,1), Re|C_{\max}$ is organized similarly to that for problem $O2|h(0, q_B), Re|C_{\max}$ given in the previous section. In fact, not only the idea of splitting the jobs into big and small is used, but also one of

the steps of Algorithm H0KB is directly included.

For finding one of the schedules our PTAS relies on greedy open shop scheduling. For a detailed discussion of greedy algorithms for the open shop problem see Section 1.8.2. Additionally, our PTAS uses a linear-time procedure that verifies whether there exists a schedule S such that for given values D_A and D_B the inequalities $C_A(S) \leq D_A$ and $C_B(S) \leq D_B$ hold simultaneously. The first version of this procedure is given in [135], and it has been significantly simplified by van den Akker et al. [4]. Shakhlevich and Strusevich [135] consider two-machine open shop problem subject to minimise an arbitrary non-decreasing non-negative function $\Phi(C_A(S), C_B(S))$. Their algorithm creates 11 heuristic schedules in linear time and at least one of them delivers the minimum to this function. Van den Akker et al. [4] formulate 4 conditions on D_A and D_B, which can be verified in linear time. These conditions are necessary and sufficient for the existence of a feasible schedule which satisfies the inequalities $C_A(S) \leq D_A$ and $C_B(S) \leq D_B$. They also describe the process of constructing such a schedule which requires linear time. Both these approaches use the algorithm of Gonzalez and Sahni which is described in Section 1.8.1.

Algorithm H11

INPUT: Problem $O2|h(1,1), Re|C_{\max}$ with $t_B \geq t_A$ and an $\varepsilon > 0$

OUTPUT: A heuristic schedule S_F

1. Run the algorithm by van den Akker et al. [4] to verify whether there exists a schedule S_F with $C_A(S_F) \leq s_A$ and $C_B(S_F) \leq s_B$. If such at schedule exists, stop; otherwise, go to Step 2.

2. Define $z = \lceil \frac{1}{\varepsilon} \rceil$ and determine the sets Z and Y of big and small jobs.

3. By full enumeration find a schedule S_z from the class $\mathcal{S}(Z)$ that is optimal for problem $O2|h(1,1), Re|C_{\max}$ with Z as the set of jobs. Assume

that this schedule is characterized by the partition $Z_{AB} \cup Z_{BA}$, and denote the corresponding sequences of jobs by $\varphi(Z_{AB})$ and $\psi(Z_{BA})$.

4. Construct schedule S_G as follows:

   (a) If no machine is idle in schedule S_z until it completes all its jobs, then denote $S'_z = S_z$ and go to Step 4(b). Otherwise, find the idle interval $[d', d'']$ either on machine A before the sequence $\psi(Z_{BA})$ or on machine B before the sequence $\varphi(Z_{AB})$. Scanning the jobs of set Y in an arbitrary sequence, assign them to be processed on the corresponding machine in the interval $[d', d'']$ one after another until the job J is found that completes after time d''. Increase the starting times of all jobs in the sequence $\psi(Z_{BA})$ (or $\varphi(Z_{AB})$, respectively) so that the first job in the sequence starts at the completion time of job J. Call the resulting partial schedule S'_z.

   (b) Construct schedule S_G by assigning the remaining jobs to be processed on machine A starting at time $C_A(S'_z)$ and on machine B at time $C_B(S'_z)$ in the greedy manner, i.e., never leaving a machine idle if there is a job ready to start processing on it.

5. Construct schedule S_H as described in Step 3 of Algorithm H0KB.

6. Determine the best of the found schedules S_G and S_H. Call this schedule S_F and stop.

We now analyze Algorithm H11.

Theorem 5.2 *For problem $O2|h(1,1)|C_{\max}$, Algorithm H11 is a polynomial approximation scheme.*

Proof. First, suppose that the algorithm stops having found schedule S_F in Step 1. Notice that in this case the algorithm by van den Akker et al. [4] guarantees that S_F has the smallest makespan. This implies that if for

problem $O2|h(1,1), Re|C_{\max}$ there exists an optimal schedule in which each machine completes its jobs before the corresponding hole, that schedule will be found and output in Step 1. Notice that Step 1 requires $O(n)$ time.

Thus, in further analysis we assume that in any optimal schedule at least one machine does not complete all jobs before the hole.

We start with schedule S_G found in Step 4 and prove that

$$\frac{C_{\max}(S_G)}{C_{\max}(S^*)} \leq 1 + \frac{1}{z} \tag{5.5}$$

under the assumption that in any optimal schedule there are jobs processed on machine B after the hole. This assumption gives rise to the lower bound

$$C_{\max}(S^*) \geq t_B \geq t_A. \tag{5.6}$$

Without of loss of generality, assume that in schedule S_z the idle interval $[d', d'']$ occurs on machine B; otherwise, the proof is symmetric. The actions described in Step 4(a) are aimed at reducing this idle period.

If the interval $[d', d'']$ is long enough to accommodate all jobs of set Y then in schedule S'_z all these jobs are completed on B earlier than time $C_A(S_z)$ and in Step 4(b) the jobs of set Y can be processed on A as a block starting at time $C_A(S_z)$. In the resulting schedule machine A is permanently busy and $C_B(S_G) = C_B(S_z) \leq C_{\max}(S^*)$, so that schedule S_G is optimal.

In the alternative situation, job J is the last job inserted on B between the sequences $\psi(Z_{BA})$ and $\varphi(Z_{AB})$, and in schedule S'_z no machine is idle.

Assume that in schedule S_G a new idle period appears on one of the machines $P \in \{A, B\}$; otherwise, this schedule is optimal. Due to the greedy nature of Step 4(b) and Lemma 1.2, only one job, say, job $J_k \in Y$, can be processed on P after the idle interval, and, moreover, during that idle interval the other machine Q is either unavailable or processes job J_k and is permanently busy till job J_k is completed. We only need to consider the case that machine P terminates schedule S_G.

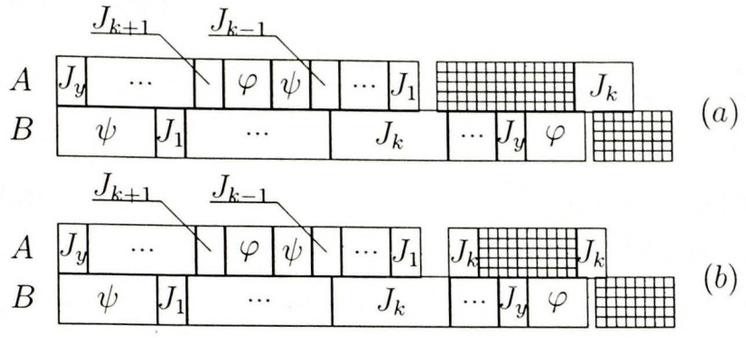

Figure 5.3: Schedule S_H with job J_k affected by the hole on machine A

If job J_k on machine P is interrupted by the hole, then $C_{\max}(S_G) \leq t_P + p_k$, where p_k is the processing time of job J_k on machine P, and (5.5) holds due to (5.2) and (5.6) .

If job J_k on machine P is not interrupted by the hole, then $C_{\max}(S_G) \leq C_Q(S_G) + p_k$. Since machine Q is permanently busy, it follows that $C_Q(S_G) \leq C_{\max}(S^*)$ and (5.5) holds due to (5.2).

We now consider schedule S_H found as described in Step 3 of Algorithm H0KB and prove that (5.4) holds under the assumption that in any optimal schedule there are no jobs processed on machine B after the hole and on machine A there are jobs processed after the hole, i.e., $t_A \leq C_{\max}(S^*)$. The proof remains identical to that of Theorem 5.1. Here, however, we need to consider two extra cases, which arise when the condition of Step 3(b) holds and job J_k on machine A is affected by the hole. It is possible that job J_k cannot start on A before the hole since it is processed on B, and starts at time t_A; see Figure 5.3(a). Alternatively, job J_k starts on A at $C_{kB}(S_H)$ and is then interrupted by the hole, so that $C_A(S_H) = C_{kB}(S_H) + \Delta_A + a_k$; see Figure 5.3(b). In either case, $C_A(S_H) \leq t_A + a_k$, so that (5.4) holds.

As in the proof of Theorem 5.1, finding a schedule S_z in Step 3 requires constant time for a fixed ε and the other steps take $O(n)$ time. Thus, Algorithm H11 is a polynomial-time approximation scheme. ■

5.5 Conclusion

This chapter describes two polynomial-time approximation schemes for two problems of scheduling jobs in a two-machine open shop subject to machine non-availability constraints under the resumable scenario. Notice that all two-machine open shop problems with a more elaborate structure of non-availability intervals are not approximable in polynomial time, unless P=NP. Our schemes rely on a popular idea of classifying jobs according to their sizes. However, we do not use linear programming or integer programming, but rather exploit combinatorial properties of the relevant schedules.

No PTAS has been previously known for shop scheduling problems with non-availability constraints, the best known algorithms provide a ratio of 4/3, and only for problems with a single hole. Being a considerable improvement, our results, however, do not resolve the approximability issue of the two-machine open shop completely. In particular, it remains unknown whether problem $O2|h(0,1),Re|C_{\max}$ admits a fully polynomial-time approximation scheme (FPTAS). For example, it is worth studying whether a dynamic programming algorithm by Lorigeon et al. [100] can be converted into an FPTAS.

If we increase the number of machines, then extending the traditional technique used by Breit [17] and Kubiak et al. [76], it is not difficult to prove that the three-machine open shop problem with one hole on each machine is not approximable in polynomial time, unless P=NP. This means that polynomial-time approximation algorithms may only exist for the extensions of our basic models $O2|h(1,1),Re|C_{\max}$ and $O2|h(0,q_B),Re|C_{\max}$ obtained by adding several continuously available machines. Using our PTAS's as well as the PTAS by Sevastianov and Woeginger [134] for the general open shop with no constraints, each of these extended problems can be approximated within a factor of $2+\varepsilon$. It is an interesting research goal to reduce this ratio bound.

Chapter 6

Scheduling machine maintenance

6.1 Introduction

The main trend in the development of deterministic Scheduling Theory has always been that of increasing the complexity and practical relevance of the models. The so-called classical models are too ideal to handle various restrictions that may occur in practical scheduling, thus, their extensions that involve additional constraints (precedence, resource, transportation, etc.) are of permanent strong interest. This chapter is based on papers [82] and [83].

We study two-machine open shop and flow shop scheduling models and concentrate on the case that each machine has to be maintained exactly once during the planning period. Additionally we present a PTAS for the two-machine flow shop problem with no-wait in process in which one of the machines is subject to maintenance. The objective for all considered problems is to minimise the makespan, i.e., the maximum completion time of the activities to be scheduled. A review of the recent literature and the considered scheduling models can be found in Section 2.5.

In this chapter, we study these three basic two-machine models and concentrate on the case that each machine has to be maintained once during the planning period, and the objective is to minimize the makespan. Un-

like the classical scheduling models, here we define the makespan not as the completion time of the last job, but as the maximum completion time of all activities to be scheduled, including the maintenance periods.

As always the case with a new problem, the issue of our primary concern will be to establish its complexity status, i.e., to find out whether the problem admits a polynomial time algorithm or is NP-hard. In the latter case, an appealing goal is to design approximation algorithms and analyze their performance.

Our study is relevant not only to the scheduling models with fixed machine non-availability intervals as discussed above, but also to the models with variable (or time-dependent) processing times. In the latter type of models, the durations of operations are not constants but depend on the start time and are represented by functions similar to $\alpha + f(t)$. The case of non-decreasing functions $f(t)$ has received special attention; the jobs of this type are normally called deteriorating. See a recent survey by Cheng et al. [29] for a literature review on scheduling with variable processing times.

The models we consider can be given an additional meaningful interpretation in terms of multi-agent scheduling recently studied by Agnetis et al. [3]. We may assume that the jobs belong to one agent and treat the maintenance periods as operations that belong to the second agent. The goal is to minimize the completion time of all jobs on all machines, provided that the processing times of the operations owned by the second agent are time-dependent.

We denote the considered problems by $F2|m(1,1)|C_{\max}$, $O2|m(1,1)|C_{\max}$, $F2|no{-}wait, m(1,0)|C_{\max}$ and $F2|no{-}wait, m(0,1)|C_{\max}$ where '$m(1,0)$' and '$m(0,1)$' in the middle field denote the fact that there is only one maintenance period (MP) on the first machine or second machine, respectively, '$m(1,1)$' denotes the fact that there is exactly one MP on each machine. Notice that, without loss of generality we may restrict our search for an optimal schedule to the class of schedules in which no processing operation is inter-

rupted by an MP, since it is always possible to start the MP earlier, right before the affected operation, without increasing the objective function.

Since in the considered problems the length and position of each MP depends on scheduling decisions, we call the resulting intervals of machine non-availability *floating*.

The remainder of this chapter is organized as follows. In Section 6.2 we demonstrate that the two-machine open shop problem with one maintenance interval on each machine is polynomially solvable for quite general functions defining the length of these intervals. By contrast, the flow shop counterpart studied in Section 6.3 is proved binary NP-hard even if the length of the maintenance interval depends linearly on its starting time. We also give a pseudopolynomial dynamic programming algorithm and two approximation algorithms, including a fully polynomial approximation scheme. The obtained results completely resolve the issues of complexity and approximation for the problems under consideration. In Section 6.4 we design and analyze a PTAS for the two-machine flow shop problem with no-wait in process. Section 6.5 contains concluding remarks.

6.2 Open Shop

In this section we consider problem $O2|m(1,1)|C_{\max}$ with a single maintenance period on each of the machines. We show that the problem of finding a schedule S^* that minimizes the makespan can be solved in $O(n)$ time in the general case that the length of an MP on machine $L \in \{A, B\}$ is equal to $\Delta_L(t) = \alpha_L + f_L(t)$, where $\alpha_L \geq 0$ and $f_L(t)$ is a non-decreasing function such that $f_L(0) = 0$, provided that computation of a function $f_L(t)$ requires at most linear time for each t. In this section, without loss of generality we assume that

$$\alpha_A \geq \alpha_B. \tag{6.1}$$

In the case that the maintenance interval is required only on one machine,

we assume that this machine is machine A, while $\alpha_B = 0$, $f_B(t) \equiv 0$.

We start with deriving global lower bounds on the length of a feasible schedule, that hold irrespective of the position of the MPs. The global *machine-based* bound

$$LB_1 = \max\{\alpha_A + a(N), \alpha_B + b(N)\} \tag{6.2}$$

holds because it is required to complete the jobs and the MP on each machine. Similarly, the global job-based bound

$$LB_2 = \max_{j \in N}\{a_j + b_j\} \tag{6.3}$$

holds due to the fact each job must be completed.

For any schedule S, the inequality

$$C_{\max}(S) \geq \max\{LB_1, LB_2\}$$

always holds, and if it holds as equality, the corresponding schedule is optimal.

In our consideration we will make use of an $O(n)$-time algorithm by Lu and Posner [101]. The algorithm of Lu and Posner finds an optimal schedule for problem $O2|h(1,0), N\text{-}Re|C_{\max}$ with a single fixed non-availability interval on machine A that starts at time zero. The latter auxiliary problem will be called Problem $R(\delta)$, provided that the length of the non-availability period on machine A is equal to δ.

Define

$$\delta = \alpha_A - \alpha_B, \tag{6.4}$$

and run the algorithm of Lu and Posner for the resulting Problem $R(\delta)$. Let S_{LP} be a found schedule that is optimal for problem $R(\delta)$.

Convert schedule S_{LP} for problem $R(\delta)$ into schedule $S^*_{0,0}$ for the original problem $O2|m(1,1)|C_{\max}$ by increasing all starting times by α_B. As will be seen later, for many instances schedule $S^*_{0,0}$ appears to be the global optimal solution, since its makespan often meets the global lower bound.

There are, however, two cases in which we may want to compare schedule $S^*_{0,0}$ with other candidate schedules, in which on one of the machines the MP starts later than time zero. The first case arises, if there exists a job r such that for problem $R(\delta)$

$$C_{\max}(S_{LP}) = a_r + b_r. \tag{6.5}$$

Schedule $S^*_{0,0}$ derived from S_{LP} need not be the optimal solution of the original problem $O2|m(1,1)|C_{\max}$, and we create two candidate schedules by starting job r at time zero on one of the machines.

To present the second case, define

$$H = \{j|\alpha_A + a_j + b_j > \alpha_B + b(N)\}. \tag{6.6}$$

If set H is not empty, define a job $p \in H$ such that

$$a_p = \min\{a_j|j \in H\} \tag{6.7}$$

and job $q \in H$ such that

$$a_q + b_q = \min\{a_j + b_j|j \in H\}. \tag{6.8}$$

For problem $R(\delta)$, there are two obvious lower bounds on the optimal makespan: the machine-based lower bound $\max\{\delta + a(N), b(N)\}$ and the job-based lower bound (6.3) As proved by Lu and Posner [101], the value of $C_{\max}(S_{LP})$ may appear to be larger than the strongest of these lower bounds. This situation arises if $|H| \geq 2$ and $b(H) + a_p > \max\{\delta + a(N), b(N)\}$, so that

$$C_{\max}(S_{LP}) = \min\{b(H) + a_p, \delta + a_q + b_q\}. \tag{6.9}$$

If this happens, then schedule $S^*_{0,0}$ derived from S_{LP} need not be the optimal solution of the original problem $O2|m(1,1)|C_{\max}$. We create two additional candidate schedules: one by processing job p immediately before the MP on machine A, and the other by processing all jobs of set H immediately before the MP on machine B. In both of these candidate schedules the MP

on the other machine still starts at time zero, and there is no need to consider the schedules in which both MP start later than zero.

The formal statement of the algorithm is given below. In the description of the algorithm $\pi(Q)$ denotes an arbitrary permutation of the jobs of a non-empty set $Q \subseteq N$; if Q is empty then $\pi(Q)$ is a dummy permutation. If a sequence $\pi(Q)$ is said to be processed on a machine $L \in \{A, B\}$, then this means that the jobs of set Q are processed as a block, one after another without intermediate idle time.

Algorithm O2

1. Given an instance of problem $O2|m(1,1)|C_{\max}$, compute δ according to (6.4) and define Problem $R(\delta)$ as the two-machine open shop problem to minimize the makespan, provided that the processing times are respectively equal to those in the original instance and machine A is not available before time δ. Run the algorithm by Lu and Posner [101] and find schedule S_{LP} that is optimal for Problem $R(\delta)$. Convert schedule S_{LP} into schedule $S^*_{0,0}$ for the original problem $O2|m(1,1)|C_{\max}$ by delaying the starting time of each job by δ.

2. If there exists a job r for which schedule S_{LP} satisfies (6.5), then go to Step 3; otherwise go to Step 4.

3. If $\alpha_B = 0$ and $f_B(t) \equiv 0$, then output schedule $S^*_{0,0}$ as the optimal schedule S^* and stop; otherwise, for the original problem $O2|m(1,1)|C_{\max}$, find the following two schedules $S^*_{0,r}$ and $S^*_{r,0}$. In schedule $S^*_{0,r}$, machine B starting at time zero processes job r, the MP on B starts at time b_r and is immediately followed by the block of jobs $\pi(N\backslash\{r\})$. The MP on machine A starts at time zero and is immediately followed by the block of jobs $\pi(N\backslash\{r\})$. Job r starts on A as early as possible, i.e., at time $\max\{\alpha_A + a(N\backslash\{r\}), b_r\}$. Schedule $S^*_{r,0}$ can be seen as the mirror image of $S^*_{0,r}$. In $S^*_{r,0}$, machine A starting at time zero processes job

r, the MP on A starts at time a_r and is immediately followed by the block of jobs $\pi(N\backslash\{r\})$. The MP on machine B starts at time zero and is immediately followed by the block of jobs $\pi(N\backslash\{r\})$. Job r starts on B as early as possible, i.e., at time $\max\{\alpha_B + b(N\backslash\{r\}), a_r\}$. Output the best of the schedules $S^*_{0,0}$, $S^*_{0,r}$ and $S^*_{r,0}$ as the optimal schedule S^* and stop.

4. Find the set H of jobs according to (6.6), and if that set is not empty, find the jobs $p \in H$ and $q \in H$ that satisfy (6.7) and (6.8), respectively. If $|H| \geq 2$ and the inequality $b(H) + a_p > \max\{\delta + a(N), b(N)\}$ holds, then go to Step 5; otherwise, output $S^* = S^*_{0,0}$ and stop.

5. For the original problem $O2|m(1,1)|C_{\max}$, find the following schedule $S^*_{p,0}$. Machine A processes job p starting at time zero. The MP on that machine starts at time a_p and is immediately followed by a sequence $\pi(N\backslash\{p\})$. On machine B the MP starts at time zero and is is immediately followed by a sequence $\pi(N\backslash\{p\})$. Job p starts on B as early as possible, i.e., at time $\max\{a_p, \alpha_B + b(N\backslash\{p\})\}$.

6. If there is the MP on machine B and if the set $\hat{H}$ of jobs such that

$$\hat{H} = \{j \in H | b(H\backslash\{j\}) \leq \alpha_A\}, \tag{6.10}$$

is not empty, then find the following schedule $S^*_{0,H}$. Determine the job u, such that

$$a_u = \min\left\{a_j | j \in \hat{H}\right\}.$$

Machine B starting at time zero processes the block of jobs $(\pi(H\backslash\{u\}), u)$. The MP on B starts at time $b(H)$ and is immediately followed by $\pi(N\backslash H)$. The MP on machine A starts at time zero and is immediately followed by the block of jobs $(\pi(N\backslash H), \pi(H\backslash\{u\}))$. Job u starts on A as early as possible, i.e., at time $\max\{\alpha_B + a(N\backslash\{u\}), b(H)\}$.

7. Output the best of the schedules found in Steps 1, 5 and 6 as the optimal schedule S^* and stop.

Since the algorithm by Lu and Posner requires $O(n)$ time, and all other steps of Algorithm O2 can be implemented in linear time, we conclude that overall running time of our algorithm is $O(n)$, provided that each of the values $f_A(a_r)$, $f_B(b_r)$, $f_A(a_p)$ and $f_B(b(H))$ can be found in at most linear time. We now prove the correctness of the algorithm.

Theorem 6.1 *Algorithm O2 finds schedule S^* that is optimal for problem $O2|m(1,1)|C_{\max}$.*

Proof. Assume first that job r satisfying (6.5) exists. This implies that

$$a_r \geq b(N\backslash\{r\}) \tag{6.11}$$

and $b_r \geq \delta + a(N\backslash\{r\})$, which is equivalent to

$$b_r + \alpha_B \geq \alpha_A + a(N\backslash\{r\}). \tag{6.12}$$

If $\alpha_B = 0$ and $f_B(t) \equiv 0$ (no MP on machine B) then $C_{\max}(S_{LP}) = C_{\max}(S^*_{0,0}) = a_r + b_r$, and this schedule is optimal for the original problem due to (6.3). Otherwise, $C_{\max}(S^*_{0,0}) = \alpha_B + a_r + b_r$. It follows that this makespan can only be reduced if job r starts before an MP on one of the machines. We create schedules $S^*_{0,r}$ and $S^*_{r,0}$ as described in Step 3.

To see that $S^*_{0,r}$ exists, notice that scheduling job r on machine A produces no clashes. The block of jobs $N\backslash\{r\}$ completes on A at time $\alpha_A + a(N\backslash\{r\})$ and starts on B at time $b_r + \alpha_B + f_B(b_r) \geq b_r + \alpha_B$, so that (6.12) ensures feasibility. In $S^*_{0,r}$, machine A completes all its work at time $\max\{\alpha_A + a(N), a_r + b_r\}$, which cannot be reduced due to (6.2) and (6.3). On the other hand, machine B completes at $b_r + \alpha_B + f_B(b_r) + b(N\backslash\{r\})$, and this time cannot be reduced in the class of schedules in which job r is processed before the MP on machine B. See Figure 6.1.

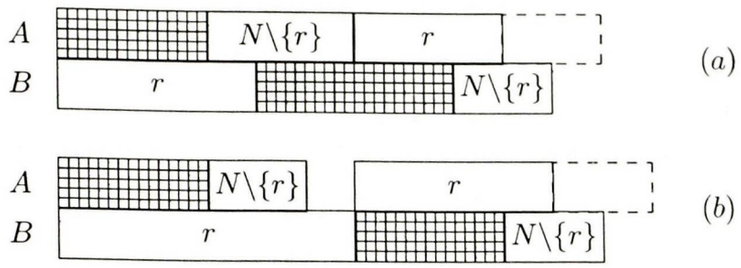

Figure 6.1: Schedule $S^*_{0,r}$

Similarly, it can be proved that schedule $S^*_{r,0}$ exists. Scheduling job r on machine B produces no clashes. The block of jobs $N\setminus\{r\}$ completes on B at time $\alpha_B + b(N\setminus\{r\})$ and starts on A at time $a_r + \alpha_A + f_A(a_r) \geq a_r + \alpha_A$, so that (6.1) and (6.11) guarantee feasibility. In $S^*_{r,0}$, machine B completes all its work at time $\max\{\alpha_B + b(N), a_r + b_r\}$, which cannot be reduced due to (6.2) and (6.3). On the other hand, machine A completes at $\alpha_A + f_A(a_r) + a(N)$, and this time cannot be reduced in the class of schedules in which job r is processed before the MP on machine A.

Algorithm O2 outputs schedule S^* such that

$$\begin{aligned} C_{\max}(S^*) = \min \quad & \{\alpha_B + a_r + b_r, \\ & \max\{\alpha_A + a(N), a_r + b_r, \alpha_B + f_B(b_r) + b(N)\}, \\ & \max\{\alpha_B + b(N), a_r + b_r, \alpha_A + f_A(a_r) + a(N)\}\} \end{aligned}$$

Analyzing the conditions in Step 4, notice that Lu and Posner [101] prove that if either $|H| \leq 1$ or $b(H) + a_p \leq \max\{\delta + a(N), b(N)\}$, then for Problem $R(\delta)$ their algorithm finds an optimal schedule S_{LP} with $C_{\max}(S_{LP}) = \max\{\delta + a(N), b(N)\}$. This schedule converts into schedule $S^*_{0,0}$ for the original problem and is optimal due to (6.2), since $C_{\max}(S^*_{0,0}) = LB_1$.

Thus, we need to consider the case that (6.9) holds, which implies that schedule S_{LP} converts into schedule $S^*_{0,0}$ for the original problem such that

$$C_{\max}(S^*_{0,0}) = \min\{\alpha_A + a_q + b_q, \alpha_B + b(H) + a_p\} > LB_1. \tag{6.13}$$

Schedule $S^*_{0,0}$ need not be the global optimal solution and the purpose of Steps 5 and 6 of our algorithm is to find a schedule with a smaller makespan.

We split our further consideration into two parts, depending on the machine on which the MP starts later than time zero.

Part A. Assume that there exists an optimal schedule in which the MP on machine B starts at time zero. If *all* jobs of set H are processed on A after the MP on that machine, then the value of $C_{\max}(S^*_{0,0})$ of the form (6.13) cannot be reduced. Indeed, if in such an optimal schedule S^* a job $j \in H$ had the processing route (A, B), then due to (6.8) we would have $C_{\max}(S^*) \geq \alpha_A + a_j + b_j \geq \alpha_A + a_q + b_q \geq C_{\max}(S^*_{0,0})$. Alternatively, if each job of set H had the processing route (B, A), then for a job $j \in H$ that was scheduled on B later than the other jobs of that set we would have due to (6.7) that $C_{\max}(S^*) \geq \alpha_B + b(H) + a_j \geq \alpha_B + b(H) + a_p \geq C_{\max}(S^*_{0,0})$.

Thus, our only hope to find a schedule better than $S^*_{0,0}$ among those schedules in which the MP on machine B starts at time zero is to search the class of schedules in which at least one job of set H is processed before the MP on machine A. Schedule $S^*_{p,0}$ belongs to that class. To see that this schedule exists, notice that scheduling job p on B produces no clashes. Besides, it follows from the definition of set H given by (6.6) that the block of jobs $N \backslash \{p\}$ starts on A no earlier than that block completes on machine B. We have that machine A completes its jobs at time $\alpha_A + f_A(a_p) + a(N)$, and this value cannot be reduced in the class of schedules under consideration due to (6.7). On the other hand, machine B completes all its work at time $\max\{\alpha_B + b(N), a_p + b_p\}$ and reaches at least one of the global lower bounds LB_1 or LB_2. It is clear that there is no advantage to delay the MP on machine B, i.e., start it later than time zero. See Figure 6.2.

Part B. Assume now there exists an optimal schedule in which the MP on machine A starts at time zero. Similarly to Part A, the value of $C_{\max}(S^*_{0,0})$ of the form (6.13) could not be reduced if in an optimal schedule a job $j \in H$ had the processing route (A, B). Thus, we focus on the situation that *each* job of

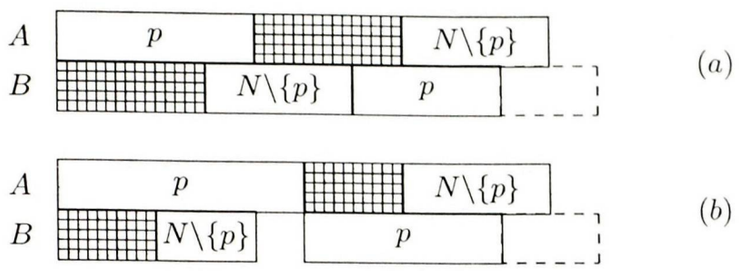

Figure 6.2: Schedule $S^*_{p,0}$

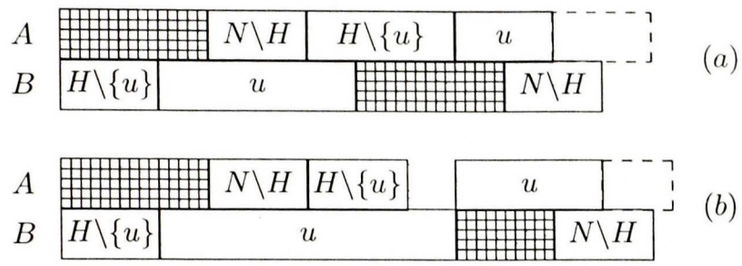

Figure 6.3: Schedule $S^*_{0,H}$

set H has the processing route (B, A). If *there exists* a non-empty subset H' of jobs of set H that are processed on B after the MP on that machine, then the value of $C_{\max}(S^*_{0,0})$ again cannot be reduced, since due to (6.7) we would have $C_{\max}(S^*) \geq b(H \backslash H') + \alpha_B + b(H') + a_j \geq \alpha_B + b(H) + a_p \geq C_{\max}(S^*_{0,0})$.

Thus, we need to search for a schedule better than $S^*_{0,0}$ among those schedules in which the MP on machine A starts at time zero and *all* jobs of set H are processed before the MP on machine B and follow the route (B, A).

If set $\hat{H}$ defined by (6.10) is empty, then in any optimal schedule $S^*$of the required structure there is a job $j \in H$ that starts on B after time α_A, so that $C_{\max}(S^*) \geq \alpha_A + a_j + b_j$, and the value of $C_{\max}(S^*_{0,0}) \leq \alpha_A + a_q + b_q$ cannot be reduced.

For a non-empty set $\hat{H}$, consider schedule $S^*_{0,H}$ found in Step 6 of the algorithm; see Figure 6.3. To see that this schedule exists, notice that scheduling job u on A produces no clashes. Further, since $u \in \hat{H}$, we deduce that the

block of jobs $H \setminus \{u\}$ completes on B no later than it starts on A. Besides, it follows from (6.13) that

$$\alpha_B + b(H) > LB_1 - a_p,$$

which due to (6.2) and (6.7) leads to

$$\alpha_B + b(H) > \alpha_A + a(N) - a_u.$$

This ensures that the block of jobs $N \setminus H$ completes on A no later than it starts on B. We have that machine B completes its jobs at time $\alpha_B + f_B(b(H)) + b(N)$, and this value cannot be reduced in the class of schedules under consideration. On the other hand, machine A completes all its work at time $\max\{\alpha_A + a(N), a_u + b(H)\}$, which cannot be reduced due to (6.2) and the choice of job u. It is clear that there is no advantage to delay the MP on machine A.

Having completed Steps 5 and 6, Algorithm O2 outputs schedule S^* such that

$$\begin{aligned} C_{\max}(S^*) = \min\{&\min\{\alpha_A + a_q + b_q, \alpha_B + b(H) + a_p\}, \\ &\max\{\alpha_A + f_A(a_p) + a(N), a_p + b_p, \alpha_B + b(N)\}, \\ &\max\{\alpha_A + a(N), a_u + b(H), \alpha_B + f_B(b(H)) + b(N)\}\}. \end{aligned}$$

This proves the theorem. ■

As can be seen from the next section, the corresponding flow shop problem is harder to solve.

6.3 Flow Shop

In this section we consider the flow shop problem $F2|m(1,1)|C_{\max}$ with a single maintenance period on each of the machines, provided that the length of an MP on machine $L \in \{A, B\}$ is equal to $\Delta_L(t) = \alpha_L + f_L(t)$, where $f_L(0) = 0$.

It is obvious that there exists an optimal schedule in which the MP on machine B starts at time zero; otherwise, the MP can be interchanged with the preceding operation and that will not increase the makespan.

Unlike its open shop counterpart, the flow shop problem under consideration is NP-hard, as stated in the following theorem.

Theorem 6.2 *Problem* $F2|m(1,0)|C_{\max}$ *with one maintenance period on machine* A *is NP-hard even if* $f_A(t) = \beta_A t$.

Proof. In order to show that problem $F2|m(1,0)|C_{\max}$ is NP-hard, we use the PARTITION problem for the reduction. This problem is defined in Section 1.2.4.

Given an arbitrary instance of PARTITION define the following instance of problem $F2|m(1,0)|C_{\max}$. There are $n = r + 3$ jobs such that

$$\begin{array}{llll} a_j = e_j, & b_j = 2e_j, & j = 1, 2, \ldots, r; \\ a_{r+1} = 0, & b_{r+1} = E; & \\ a_{r+2} = 2E, & b_{r+2} = 0, & \\ a_{r+3} = 2E, & b_{r+3} = 5E. & \end{array}$$

The length of the MP on machine A is equal to $\Delta(t) = E + t$, where t is the starting time of the MP.

We show that for the constructed instance a schedule S_0 such that $C_{\max}(S_0) \leq 10E$ exists if and only if PARTITION has a solution.

Suppose that set R_1 and R_2 form a solution to PARTITION. Schedule S_0 exists and can be constructed as follows. Each machine starts at time zero and processes the jobs in the sequence $(r + 1, R_1, r + 3, R_2, r + 2)$, where the jobs of sets R_1 and R_2 are scheduled in any order. The MP on machine A starts after job $r + 3$ at time $3E$, so that its length is $4E$.

Suppose now that schedule S_0 exists. Since the total workload on machine B is equal to $10E$, it follows that B is permanently busy in the time interval $[0, 10E]$. Thus, job $r + 1$ must be the first in the processing sequence. The MP cannot start on A at time zero, since this generates idle time on B after

job $r+1$. Due to the same reason neither job $r+2$ nor job $r+3$ can start on A at time zero. Thus, machine A after job $r+1$ processes a sequence σ of jobs of set R. Let X be the total processing time of the jobs of that sequence on machine A. If sequence σ is immediately followed by the MP, then the MP completes at time $X+(E+X)$, and the last job of sequence σ completes on B at time $E+2X$. This implies that none of the remaining jobs can start on B at time $E+2X$, which is impossible.

If sequence σ is followed by job $r+2$, this also generates idle time on B after time $E+2X$. Thus, sequence σ must be followed by job $r+3$. If $X<E$, then the last job of sequence σ completes on B earlier that job $r+3$ completes on A, thereby generating idle time on B. We deduce that $X \geq E$.

The MP completes at time $X+2E+(3E+X)=5E+2X$ and total processing time of the remaining jobs on machine A is equal to $(2E-X)+2E$, so that the last job is completed on A at time $9E+X$. Since $10E = C_{\max}(S_0) \geq 9E+X$ and $X \geq E$, we derive that $X=E$. Thus, the set of jobs in sequence σ and the set of the remaining jobs of set R form a solution to PARTITION. ■

We resolve the exact complexity status of problem $F2|m(1,1)|C_{\max}$ by providing a dynamic programming algorithm for its solution. The running time of the algorithm is pseudopolynomial with respect to the length of input, provided that

- all processing times and values α_A and α_B are integers;
- function $f_A(t)$ has integer values for all integer times t, $0 \leq t \leq a(N)$.

Consider an optimal schedule for problem $F2|m(1,1)|C_{\max}$ in which the MP on machine A starts at time t_0. Without loss of generality, we may assume that the jobs processed on machine A in the time interval $[0,t_0]$ are sequenced according to Johnson's rule; otherwise these jobs can be rearranged accordingly without increasing the makespan. Due to a similar reason, the

jobs that follow the MP on machine A are also ordered according to the Johnson's rule. Notice that a similar property is observed for the two-machine flow shop problem with a fixed non-availability interval, see Lee [89].

Without loss of generality, assume that all jobs of set N are numbered according to Johnson's rule. Our algorithm for problem $F2|m(1,1)|C_{\max}$ scans the jobs in the order of their numbering and builds two (partial) flow shop schedules. One of them, which we call *left*, handles the jobs that are processed before the MP on machine A, provided that their processing starts at time zero and the MP on machine B also starts at zero. The other one, which we call *right*, handles the jobs to be processed after the MP on machine A, provided that their processing starts at time zero. Once each job is assigned to one of these schedules, either left or right, the two schedules can be concatenated together with the MP on machine A of an appropriate length placed between them. As a result, a feasible schedule for the original problem is constructed.

Suppose k jobs have been assigned to the left or right schedule. A typical state is described by the string

$$(k; u_1, u_2; v_1, v_2)\,, \tag{6.14}$$

where

k – the number of scheduled jobs, $0 \leq k \leq n$;

u_1 – total processing time of the jobs in the current left schedule on machine A;

u_2 – the makespan of the current left schedule;

v_1 – total processing time of the jobs in the right schedule on machine B;

v_2 – the makespan of the current right schedule;

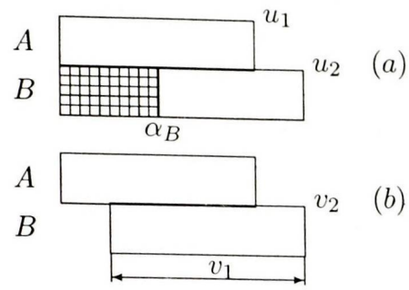

Figure 6.4: A partial schedule: (a) left schedule; (b) right schedule

These values are illustrated in partial schedules shown in Figure 6.4.

At the initialization stage, we define

$$(0; 0, \alpha_B; 0, 0)\,,$$

and scan the jobs in accordance with their numbering. Define $A_0 = 0$ and compute the partial sums $A_k = \sum_{j=1}^{k} a_j$ for all $k = 1, \ldots, n$.

Given a partial schedule associated with state (6.14), the algorithm generates two new states, depending on whether the next job $k+1$ is inserted into the left schedule or the right schedule.

If job $k+1$ is assigned as the last job of the left schedule then either this job becomes critical so that the makespan of the left schedule is equal to $u_1 + a_{k+1} + b_{k+1}$, or the critical job remains the same, i.e., the makespan increases by b_{k+1}. Thus, the new state is described by

$$(k+1; u_1 + a_{k+1}, \max\{u_1 + a_{k+1}, u_2\} + b_{k+1}; v_1, v_2)\,.$$

Otherwise, if job $k+1$ is assigned as the last job of the right schedule then the new state is described by

$$(k+1; u_1, u_2; v_1 + b_{k+1}, \max\{A_{k+1} - u_1, v_2\} + b_{k+1})\,.$$

In any case, if the resulting state coincides with an existing state, it is discarded, i.e., only one state corresponds to the same values of k, u_1, u_2, v_1 and v_2.

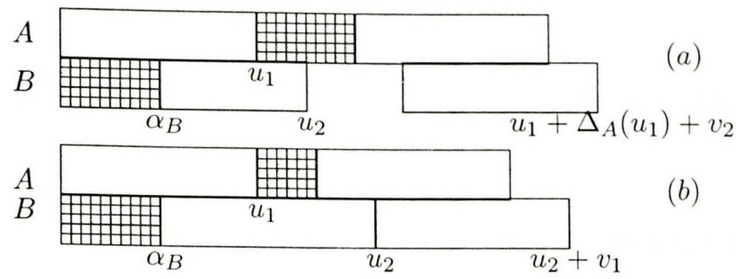

Figure 6.5: A feasible schedule S : (a) a critical job in the right schedule; (b) a critical job in the left schedule

The process is continued until all jobs are assigned. The left and right schedules related to a final state

$$(n; u_1, u_2; v_1, v_2)$$

are converted into a feasible schedule S for the original problem by placing the MP on machine A of length $\Delta_A(u_1)$ between them. The resulting makespan depends on the position of a critical job, i.e., whether it is located in the left schedule or in the right schedule. It follows that

$$C_{\max}(S) = \max\{u_2 + v_1, u_1 + \Delta_A(u_1) + v_2\}, \tag{6.15}$$

as illustrated in Figure 6.5.

The optimal makespan can be found by minimizing (6.15) over all final states. The sequence of jobs in the corresponding optimal schedule can be found by backtracking. It follows from (6.15) that all four state variables u_1, u_2, v_1 and v_2 are essential, and no information is lost by keeping only one state corresponding to their equal values.

Besides, the number of different values of u_1 and u_2 does not exceed $a(N)$ and $\max\{\alpha_B, a(N)\} + b(N)$, respectively, while the number of different values of v_1 and v_2 does not exceed $b(N)$ and $a(N) + b(N)$. Assuming $M = \max\{a(N), \alpha_B + b(N)\}$ we finally derive that the total running time of the scheme is $O(nM^4)$.

Thus, we have proved the following statement.

Theorem 6.3 *Problem* $F2\,|m(1,1)|\,C_{\max}$ *admits a pseudopolynomial time algorithm.*

We now convert the pseudopolynomial dynamic programming algorithm above into a FPTAS for problem $F2\,|m(1,1)|\,C_{\max}$, provided that the length of the MP on machine A is described by a linear function.

Consider problem $F2\,|m(1,1)|\,C_{\max}$ with the processing times a_j and b_j in which the length of an MP on machine L is equal to $\Delta_L(t) = \alpha_L + f_L(t)$, $L \in \{A, B\}$, provided that $f_A(t) = \beta_A t$. We refer to this problem as Problem P. To develop a FPTAS, we use the well–known rounding technique. Given an instance of Problem P and an $\varepsilon > 0$, define $\delta = \varepsilon M/(n(\beta_A + 3) + 1)$. Introduce Problem $\widetilde{P}$ as problem $F2|m(1,1), Re|C_{\max}$ with the processing times defined as

$$\widetilde{a}_j = \lfloor a_j/\delta \rfloor, \widetilde{b}_j = \lfloor b_j/\delta \rfloor, j = 1, 2, \ldots, n, \tag{6.16}$$

and

$$\widetilde{a}_A = \lfloor \alpha_A/\delta \rfloor, \ \widetilde{\alpha}_B = \lfloor \alpha_B/\delta \rfloor. \tag{6.17}$$

Here $\lfloor x \rfloor$ denotes the largest integer that does not exceed x.

Algorithm FP

1. Given an instance of Problem P and an $\varepsilon > 0$, define the instance of Problem $\widetilde{P}$ by (6.16) and (6.17).

2. For Problem $\widetilde{P}$, run the dynamic programming algorithm. Call the found schedule and the associated permutation of jobs indices by $\widetilde{S}$ and π, respectively. Let the MP in schedule $\widetilde{S}$ be positioned after the $v-$th job in permutation π.

3. Process the jobs from the original instance of Problem P according to the permutation π, provided that each operation starts as early as possible, and the MP on machine A starts after the first v jobs. Call the resulting schedule S_ε. Stop.

Theorem 6.4 *For problem* $F2\,|m(1,1)|\,C_{\max}$, *Algorithm FP is a FPTAS, provided that function* $f_A(t)$ *is linear.*

Proof. Given an instance of Problem P, introduce problem $F2|m(1,1)|C_{\max}$ with the processing times $\overline{a}_j$ and $\overline{b}_j$ defined as

$$\overline{a}_j = \delta\widetilde{a}_j, \overline{b}_j = \delta\widetilde{b}_j,\ j = 1, 2, \ldots, n, \tag{6.18}$$

and with

$$\overline{\alpha}_A = \delta\widetilde{\alpha}_A,\ \overline{\alpha}_B = \delta\widetilde{\alpha}_B, \tag{6.19}$$

and call this Problem $\overline{P}$.

Let π be a permutation of jobs that defines schedule $\widetilde{S}$ found in Step 2 of Algorithm FP. Due to (6.18) and (6.19) we derive that a schedule $\overline{S}$ that is optimal for Problem $\overline{P}$ is also associated with the same permutation. Since $f_A(t)$ is linear, we derive that $C_{\max}(\overline{S})$ for Problem $\overline{P}$ is δ times the makespan $C_{\max}(\widetilde{S})$ for Problem $\widetilde{P}$. Without loss of generality, we assume that the jobs are renumbered in such a way that $\pi = (1, 2, \ldots, n)$.

Recall that the processing times a_j for Problem P are obtained by extending the times $\overline{a}_j$ to their original values by no more than δ each. The same holds for the values of b_j, α_A and α_B. The total increase of durations of all $2n$ operations does not exceed $2n\delta$. Additionally, in the worst case the MP on machine A in schedule S_ε can be delayed by at most $n\delta$ time units compared to its starting time in schedule $\overline{S}$, so that the increase of the completion time of the MP on A does not exceed $n\delta + \delta + \beta_A n\delta$. This implies that

$$C_{\max}(S_\varepsilon) \leq C_{\max}(\overline{S}) + \delta\,(3n + 1 + \beta_A n)\,.$$

This due to the definition of δ yields

$$C_{\max}(S_\varepsilon) \leq C_{\max}(\overline{S}) + \varepsilon M.$$

Since each M and $C_{\max}(\overline{S})$ are lower bounds on the optimal makespan for the original Problem P, we deduce that

$$\frac{C_{\max}(S_\varepsilon)}{C_{\max}(S^*)} \leq 1 + \varepsilon.$$

The running time of Algorithm FP is determined by the running time of the dynamic programming algorithm used in Step 2. In our case the dynamic programming algorithm takes $O\left(n\widetilde{M}^4\right)$ time, where $\widetilde{M} = \max\left\{\widetilde{a}(N), \widetilde{\alpha}_B + \widetilde{b}(N)\right\}$. The definition of δ implies that $\widetilde{M} = O(n/\varepsilon)$. Thus, we conclude that the running time of Algorithm FP does not exceed $O(n^5/\varepsilon^4)$, and the algorithm is a fully polynomial approximation scheme. ■

The running time of our FPTAS coincides with that of a recent approximation scheme by Ng and Kovalyov [107] for the two-machine flow shop scheduling problem with a single fixed non-availability interval under the resumable scenario. Recall that in the latter settings, the problem with one non-availability interval on each machine is not approximable within a constant factor unless P=NP.

The running time of our FPTAS is quite large. We therefore describe an algorithm that requires only $O(n \log n)$ time and finds a schedule with the makespan that is at most 3/2 times the optimum value, provided that the function $f_A(t)$ is linear, i.e., $f_A(t) = \alpha_A + \beta_A t$.

Let S^* be an optimal schedule. The following lower bounds on the optimal makespan obviously hold:

$$C_{\max}(S^*) \geq \alpha_A + a(N), \tag{6.20}$$

$$C_{\max}(S^*) \geq \alpha_B + b(N), \tag{6.21}$$

$$C_{\max}(S^*) \geq C_{\max}(S_J^*), \tag{6.22}$$

where S_J^* is the schedule obtained by Johnson's algorithm with both MP ignored.

Define the sets of jobs

$$N_1 = \{j \in N | (1+\beta_A)a_j \leq b_j\}, \ N_2 = N \backslash N_1.$$

We prove an additional lower bound

$$C_{\max}(S^*) \geq (1+\beta_A)a(N_1) + \alpha_A + b(N_2). \tag{6.23}$$

To see that (6.23) holds, consider an optimal schedule S^* and assume that X_1 is the set of jobs scheduled on A before the MP, while X_2 is the set of jobs that are processed on A after the MP. It is clear that the MP starts at time $a(X_1)$ and the jobs of set X_2 cannot start on B earlier than time $a(X_1) + \alpha_A + \beta_A a(X_1)$, the completion time of the MP on A. Thus,

$$C_{\max}(S^*) \geq (1+\beta_A)a(X_1) + \alpha_A + b(X_2),$$

so that (6.23) holds, since by definition $(1+\beta_A)a(N_1) + \alpha_A + b(N_2) \leq (1+\beta_A)a(X_1) + \alpha_A + b(X_2)$ for any partition of set N into two subsets X_1 and X_2.

Our approximation algorithm finds two schedules and outputs the best of them as a heuristic solution. Formally it can be described as follows.

Algorithm F2H

1. Create a schedule S_1 in which both MP start at time 0, and the jobs are sequenced according to Johnson's algorithm.

2. Create a schedule S_2 in which on each machine an arbitrary sequence of jobs of set N_1 is followed by an arbitrary sequence of jobs of set N_2. The MP on B starts at time zero, while on A the MP starts right after the last job of set N_1 is completed.

3. Call the best of the found schedules S_H and output S_H.

It is obvious that the running time of Algorithm F2H is determined by the time complexity of Johnson's algorithm used in Step 1. Thus, the algorithm requires at most $O(n \log n)$ time. Below we analyze its worst-case performance.

Theorem 6.5 *For problem* $F2|m(1,1)|C_{\max}$ *with a linear function* $f_A(t)$ *Algorithm F2H finds a schedule* S_H *such that*

$$\frac{C_{\max}(S_H)}{C_{\max}(S^*)} \leq \frac{3}{2}, \tag{6.24}$$

and this bound is tight.

Proof. Throughout the proof, we assume that in each schedule S_1 or S_2 there is idle time on machine B; otherwise $C_{\max}(S_H) = \alpha_B + b(N)$, so that S_H is optimal due to (6.21).

Take schedule S_1. It follows that

$$C_{\max}(S_1) \leq \alpha_A + C_{\max}(S_J^*),$$

and in the remainder of this proof we assume that $\alpha_A > \frac{1}{2}C_{\max}(S^*)$; otherwise due to (6.22) we obtain that the theorem holds for $S_H = S_1$. This along with (6.20) implies that

$$a(N) \leq \frac{1}{2}C_{\max}(S^*). \tag{6.25}$$

Take schedule S_2. Without loss of generality, we may assume that in S_2 the jobs on machine B are partitioned into two blocks N_1 and N_2, each processed without intermediate idle time and with no clashes with the same block of jobs on machine A. It follows that

$$C_{\max}(S_2) \leq \max\left\{a(N_1) + b(N), (1+\beta_A)a(N_1) + \alpha_A + a(N_2) + b(N_2)\right\}.$$

Using (6.21) and (6.23) we derive

$$C_{\max}(S_2) \leq C_{\max}(S^*) + \max\left\{a(N_1), a(N_2)\right\},$$

which due to (6.25) yields (6.24) for $S_H = S_2$.

To see that the bound is tight, consider the following instance of problem $F2|m(1,1)|C_{\max}$. There is one job with both operations of unit length. The length of the MP on machine B is equal to 1. The length of the MP on machine A is given by the function $1 + \varepsilon t$, where $\varepsilon > 0$ is a small number. For schedule S_1 we have that the MP on machine A starts at time zero, so that $C_{\max}(S_1) = 3$. Besides, since $\varepsilon > 0$, we have that $N_1 = \varnothing$ and $N_2 = \{1\}$, so that schedule S_2 coincides with S_1. On the other hand, in the optimal schedule S^*, the MP on machine A is scheduled after the job, so that $C_{\max}(S^*) = 1 + (1 + \varepsilon) = 2 + \varepsilon$. As ε tends to zero, the ratio $C_{\max}(S_H)/C_{\max}(S^*)$ goes to 3/2. ■

Theorem 6.5 shows that the makespan of a schedule found by Algorithm F2H may happen to be 50% worse than the optimal value. However, in practice Algorithm F2H exhibits a much more accurate performance.

We have conducted computational experiments to test the behaviour of the algorithm on simulated data. The purpose of the first experiment has been to track the performance ratio depending on the number of jobs. For each value n of the number of jobs 100, 200, 500 and 1000 we generated 100 instances. The processing times of the jobs have been drawn from the uniform distribution over the interval $[1, 100]$. The lengths of the MPs on machine $L \in \{A, B\}$ have been defined by linear functions $\Delta_L(t) = \alpha_L + \beta_L t$ where α_L and β_L have been drawn from the uniform distribution over the interval $[1, 1000]$ and $[0, 1]$, respectively. For each generated instance the performance ratio has been computed as the ratio of the makespan found by Algorithm F2H over the strongest of the lower bounds (6.20)-(6.22).

The results of this experiment are shown in Table 6.1. We see that in almost 80% of instances the algorithm finds a global optimum solution. The average relative error is less than 1%. The worst-case relative error is never larger than 7%. The performance of the algorithm improves as the number of jobs grows.

n	% Optimum Found	Average Error	Maximum Error
100	62	1.0084	1.0616
200	75	1.0036	1.0486
500	83	1.0018	1.0307
1000	77	1.0008	1.0144

Table 6.1: Results of Experiment 1

	$\beta_A = 0.01$	$\beta_A = 0.1$	$\beta_A = 1$	$\beta_A = 10$
$\alpha_A = p_{\max}$	1.0001	1.0001	1.0001	1.0001
$\alpha_A = 10p_{\max}$	1.0014	1.0125	1.0291	1.0218
$\alpha_A = 100p_{\max}$	1.0007	1.0039	1.0079	1.0069

Table 6.2: Results of Experiment 2

The second experiment has been carried out to verify the performance of the algorithm for various values of parameters α_A and β_A that define the length of the MP on machine A. Recall that the MP on machine B starts at time zero in any optimal schedule. We have tested the algorithm for twelve combinations of the values of α_A and β_A, and for each combination 100 instances of $n = 100$ jobs have been generated. Table 6.2 reports the performance ratio values. Here $p_{\max}$ denotes the largest processing time.

The results show the robust performance of the algorithm. The most difficult instances correspond to the medium values of α_A and β_A. This is due to the fact that for these values a heuristic schedule is likely to position the MP on machine A in the middle part, while the lower bounds assume the MPs placed in the beginning of the schedule.

The algorithm has been coded in C++ and run on the Pentium IV 2.8GHz workstation. We do not report the computation time since the two experiments have taken several milliseconds to complete.

6.4 Flow Shop No-Wait

For problem $F2|no-wait, m(1,0)|C_{\max}$, the length of the MP that starts on A at time t is defined as $\Delta(t) = \alpha + f(t)$, where α is a given positive constant

and f is a non-decreasing function such that $f(0) = 0$ and its computation for a given t takes constant time.

To demonstrate the difficulty of problem $F2|no-wait, m(1,0)|C_{\max}$ we present an instance of the problem with only two jobs. The example demonstrates that (i) an optimal sequence of jobs may be different from an optimal sequence for the related problem $F2|no-wait|C_{\max}$ with no maintenance period and (ii) in an optimal schedule the maintenance period can be not in the beginning of the schedule where its length is minimal but in the middle of the schedule to fill a suitable idle period on machine A. Consider problem $F2|no-wait, m(1,0)|C_{\max}$ with two jobs with the processing times

$$a_1 = 1, b_1 = 2;\ a_2 = 2, b_2 = 9.$$

The length of the MP that starts on machine A at time t is given by $\Delta(t) = 2 + 3t$. Figures 6.6(a-c) show the possible placements of the MP, provided that the jobs are kept in the sequence that is optimal for problem $F2|no-wait|C_{\max}$ with the MP ignored. The schedule in Figure 6.6(d) is a unique optimal schedule for the instance of problem $F2|no-wait, m(1,0)|C_{\max}$ under consideration.

The Gilmore-Gomory algorithm is used as a subroutine in our PTAS for problem $F2|no-wait, m(1,0)|C_{\max}$. This algorithm is discussed in details in Section 1.6.3.

For our further purposes, we need the following statement.

Lemma 6.1 *Suppose that a solution to some instance of problem $F2|no-wait|C_{\max}$ is found. If now some subset Q of jobs is removed from the instance, an optimal solution for the remaining jobs can be found in $O(n)$ time.*

Proof. Recall that the Gilmore-Gomory algorithm consists of two parts: first it solves the assignment problem (matching) with the original matrix, and then merges the obtained partial tours into a complete optimal tour (patching). The matching part of the algorithm is sorting of each array $(\alpha_1, \ldots, \alpha_n)$

and $(\beta_1, \ldots, \beta_n)$ in non-decreasing order. This requires $O(n \log n)$ time. All patching steps can be implemented in linear time; see, e.g., [21]. If the jobs of set Q are removed, to solve the matching subproblem it suffices to remove these jobs from the sorted arrays available after a solution with the full set of jobs is found. The patching part will require $O(n - |Q|)$ time. ■

Remark 6.1 *We have not been able to establish the exact complexity status of problem* $F2|no-wait, m(1,0)|C_{\max}$. *We have developed our approximation scheme assuming that the problem is NP-hard, as is NP-hard its counterpart with a single fixed non-availability interval. If our problem had admitted a polynomial-time algorithm, that algorithm would have been able to solve a generalization of the Gilmore-Gomory TSP with some distances of variable length. The latter seems highly unlikely; see a recent survey on solvable cases of the TSP [21].*

Given an instance of problem $F2|no - wait, m(1,0)|C_{\max}$, let problem $F2|no - wait|C_{\max}$ with the same set of jobs and continuously available machines be called the associated Problem GG (for Gilmore-Gomory).

Without loss of generality, we may assume that in any schedule that is optimal for problem $F2|no - wait, m(1,0)|C_{\max}$ the maintenance period on machine A starts later than time zero; otherwise, such a schedule can be found by starting an optimal schedule for the associated Problem GG at time α.

Suppose that the jobs $J_1, \ldots, J_n$ are numbered in the order of their processing in schedule S^* that is optimal for problem $F2|no - wait, m(1,0)|C_{\max}$, and job J_p is the job that immediately precedes the MP. During the MP on machine A, machine B processes only operation $O_{p,B}$. If job J_p and the MP are removed from S^* then the starting times of all operations that follow the MP in S^* can be reduced by at least α. Thus, there exists a flow shop no-wait schedule S' for the remaining jobs and continuously available machines such that $C_{\max}(S') \leq C_{\max}(S^*) - \alpha$. Given a solution to

the associated Problem GG with the full set of jobs (in fact, only solution to the matching subproblem is needed; see Remark 6.1), we can find schedule S' in $O(n)$ time.

This argument can be extended to a removal of more than one job along with the MP from an optimal schedule. Given an optimal schedule S^*, consider the sequence of jobs $J_{p-k}, \ldots, J_p, \ldots, J_{p+l}$ for $k \geq 0$ and $l \geq 0$ such that the MP on machine A is placed after job J_p. We call the sequence $J_{p-k}, \ldots, J_p, MP, \ldots, J_{p+l}$ a *block* of schedule S^* and denote it by $\sigma^*(Q, MP)$, where $Q = \{J_{p-k}, \ldots, J_p, \ldots, J_{p+l}\}$; see Figure 6.7. Notice that the case that $l = 0$ corresponds to the situation that the MP is the last element of the block.

We call the sequence $\sigma^*(Q', MP)$ the *interior* of block $\sigma^*(Q, MP)$ if $Q' = \{J_{p-k+1}, \ldots, J_p, \ldots, J_{p+l-1}\}$. The length of the time interval that starts at time $R_{p-k+1,A}(S^*)$ and completes either at the end of the MP (if $l = 0$) or at time $\max\{C_{MP,A}(S^*), C_{p,B}(S^*)\}$ (if $l = 1$) or at time $C_{p+l-1,A}(S^*)$ (if $l > 1$) defines the length I_Q of the interior of block $\sigma^*(Q, MP)$. Notice, that in schedule S^*, the elements of the interior of a block can be processed simultaneously only with the elements of that block. If block $\sigma^*(Q, MP)$ is removed from S^*, then the starting times of all operations that follow the block can be reduced by at least I_Q.

Lemma 6.2 *If block $\sigma^*(Q, MP)$ is removed from schedule S^* that is optimal for problem $F2|no-wait, m(1,0)|C_{\max}$, the starting times of all operations in schedule S^* that follow the block can be reduced by at least I_Q. There exists a flow shop no-wait schedule S' for the remaining jobs and continuously available machines such that $C_{\max}(S') \leq C_{\max}(S^*) - I_Q$. Given a solution to the associated Problem GG with the full set of jobs, schedule S' can be found in $O(n)$ time.*

Proof. This lemma immidiately follows from Lemma 6.1 and the definition of the interior of a block. ■

For problem $F2|no-wait, m(1,0)|C_{\max}$ the following lower bound on the makespan of an optimal schedule

$$C_{\max}(S^*) \geq C_{\max}(S^*_{GG}) \tag{6.26}$$

obviously holds, where S^*_{GG} is an optimal schedule for the associated Problem GG.

Assume that $\varepsilon < 1$ and the number of jobs is sufficiently large, e.g., $n > 2\left\lceil \frac{1}{\varepsilon} \right\rceil + 3$.

Algorithm FNWA

1. Given an instance of problem $F2|no-wait, m(1,0)|C_{\max}$ and $\varepsilon > 0$, define z equal to $\left\lceil \frac{1}{\varepsilon} \right\rceil$.

2. Assign the MP to start at time zero and append all jobs in the sequence that defines schedule S^*_{GG}. Denote the obtained schedule S_0.

3. For each k taking odd values from 1 to $2z-3$ do the following:

   Enumerate all possibilities of a selection of k jobs from set N. For each selected set N_k do the following:

   (a) By enumerating all possibilities, solve an auxiliary problem $F2|no-wait, m(1,0)|C_{\max}$ with the set of jobs N_k. Call the obtained schedule $\hat{S}_{N_k}$.

   (b) Solve to optimality problem $F2|no-wait|C_{\max}$ for the set $N \backslash N_k$ of the remaining jobs by the Gilmore-Gomory algorithm. Call the obtained schedule $\tilde{S}_{N \backslash N_k}$.

   (c) Find schedule S_{N_k} for the original problem by concatenating schedules $\hat{S}_{N_k}$ and $\tilde{S}_{N \backslash N_k}$.

4. Among all found schedules output schedule S_H with the minimum makespan.

Let us estimate the running time of Algorithm FNWA. According to Remark 6.1, we can perform the matching part of the Gilmore-Gomory algorithm beforehand, hence running this algorithm in Step 3b requires only linear time for each set N_k. It is clear that the time required for the last run of the loop in Step 3 for $k = 2z - 3$ determines the overall time complexity of the algorithm. There are $O(n^{2z-3})$ options of selecting the set N_{2z-3}, and for each selection Steps 3a-3c can be implemented in linear time. Hence, the total running time of Algorithm FNWA does not exceed $O(n^{2\lceil \frac{1}{\varepsilon} \rceil - 2})$.

We now analyze the worst-case performance of Algorithm FNWA.

Theorem 6.6 *For a given $\varepsilon \geq 0$ Algorithm FNWA outputs a schedule S_H such that the inequality*

$$\frac{C_{\max}(S_H)}{C_{\max}(S^*)} \leq 1 + \varepsilon$$

holds for any instance of problem $F2|no-wait, m(1,0)|C_{\max}$, where S^* is an optimal schedule for the instance.

Proof. If $\alpha \leq \frac{1}{z} C_{\max}(S^*)$ then clearly $C_{\max}(S_0) \leq (1 + \frac{1}{z}) C_{\max}(S^*)$ due to (6.26). Therefore, further we concentrate only on the case that $\alpha > \frac{1}{z} C_{\max}(S^*)$ and $C_{\max}(S_0) > (1 + \frac{1}{z}) C_{\max}(S^*)$.

Since $C_{\max}(S_0) > (1+\frac{1}{z}) C_{\max}(S^*)$ we conclude that in any optimal schedule at least one job is processed before the MP.

In the first iteration of the loop in Step 3 for $k = 1$, any selected set N_1 consists of exactly one job. Schedule $\hat{S}_{N_1}$ is an optimal flow shop no-wait schedule for processing the job in N_1 together with the MP. Schedule $\tilde{S}_{N \setminus N_1}$ is an optimal flow shop no-wait schedule of the jobs of set $N \setminus N_1$. If these two schedules are concatenated, we obtain schedule S_{N_1} such that $C_{\max}(S_{N_1}) \leq C_{\max}(\hat{S}_{N_1}) + C_{\max}\left(\tilde{S}_{N \setminus N_1}\right)$. Due to full enumeration, among the generated sets N_1 there will be set $\bar{N}_1$ that consists of a job that is located immediately before the MP in a certain optimal schedule S^*, i.e., $\sigma^*(\bar{N}_1, MP)$ is a block of schedule S^*. Due to Lemma 6.2 applied for $k = 0$

and $l = 0$, when this block is removed from schedule S^* the makespan of that schedule decreases by at least α. Thus, for $N_1 = \bar{N}_1$ we derive that $C_{\max}\left(\tilde{S}_{N\setminus N_1}\right) \leq C_{\max}(S^*) - \alpha \leq \left(1 - \frac{1}{z}\right) C_{\max}(S^*)$. In the reminder of this proof we assume that $C_{\max}\left(\hat{S}_{N_1}\right) > \frac{2}{z} C_{\max}(S^*)$ for $N_1 = \bar{N}_1$; otherwise the theorem holds for $S_H = S_{\bar{N}_1}$.

Notice that in the sequence of jobs that defines an optimal schedule S^* one of the following three configurations are possible:

(i) there is at least one job located immediately before block $\sigma^*(\bar{N}_1, MP)$ and one job located immediately after;

(ii) there are no jobs located before block $\sigma^*(\bar{N}_1, MP)$;

(iii) there are no jobs located after block $\sigma^*(\bar{N}_1, MP)$.

Consider the next iteration of the loop in Step 3. Among the sets N_3 generated in this iteration there will be a set, which we call $\bar{N}_3$, such that $\bar{N}_3 = \bar{N}_1 \cup \{J_u, J_v\}$ and, moreover, in the sequence that defines an optimal schedule S^* job J_u is processed immediately before block $\sigma^*(\bar{N}_1, MP)$ while job J_v immediately follows block $\sigma^*(\bar{N}_1, MP)$, provided that schedule S^* admits configuration (a) above. In the case of configuration (b), both J_u and J_v are sequenced in S^* immediately after block $\sigma^*(\bar{N}_1, MP)$, and in the case of configuration (c), the jobs J_u and J_v are sequenced immediately before block $\sigma^*(\bar{N}_1, MP)$. In other words, the algorithm finds set $\bar{N}_3 \supset \bar{N}_1$ of three jobs and such that $\sigma^*(\bar{N}_3, MP)$ is a block of an optimal schedule S^*. Moreover, in schedule S^* block $\sigma^*(\bar{N}_1, MP)$ belongs to the interior of block $\sigma^*(\bar{N}_3, MP)$.

If now block $\sigma^*(\bar{N}_3, MP)$ is removed from S^*, the starting time of all jobs that follow the block in schedule S^* can be reduced by at least the length of its interior, which in turn is no smaller than $C_{\max}\left(\hat{S}_{\bar{N}_1}\right) > \frac{2}{z} C_{\max}(S^*)$. Due to Lemma 6.2, this implies that $C_{\max}\left(\tilde{S}_{N\setminus N_3}\right) \leq \left(1 - \frac{2}{z}\right) C_{\max}(S^*)$ for $N_3 = \bar{N}_3$.

In the reminder of this proof we assume that $C_{\max}\left(\hat{S}_{N_3}\right) > \frac{3}{z}C_{\max}(S^*)$ for $N_3 = \bar{N}_3$; otherwise the theorem holds for $S_H = S_{\bar{N}_3}$.

Extending this argument, we can prove that for any odd $k, 1 \leq k \leq 2z-3$, either $C_{\max}\left(\hat{S}_{N_k}\right) > \frac{k+3}{2z}C_{\max}(S^*)$ for some set $N_k = \bar{N}_k \supset \bar{N}_{k-1} \supset \ldots \supset \bar{N}_1$ or the theorem holds for $S_H = S_{\bar{N}_k}$. However, for $k = 2z-3$ we obtain $C_{\max}\left(\hat{S}_{\bar{N}_k}\right) > C_{\max}(S^*)$ which is impossible.

This proves that the algorithm generates at least one schedule which satisfies the inequality $C_{\max}(S_H) \leq (1+\frac{1}{z})C_{\max}(S^*) \leq (1+\varepsilon)C_{\max}(S^*)$ according to the definition of z. ∎

A PTAS for problem $F2|no-wait, m(0,1)|C_{\max}$ with the MP on machine B can be obtained from Algorithm FNWA by making simple symmetric changes, e.g., replacing the references to problem $F2|no-wait, m(1,0)|C_{\max}$ in Step 1 and Step 3a by $F2|no-wait, m(0,1)|C_{\max}$. The removal argument and Lemma 6.2 can be modified accordingly.

6.5 Conclusion

We have studied the two-machine open shop, flow shop and flow shop no-wait problems to minimise the makespan in which machines are subject to preventive maintenance, and the length of each maintenance period depends on its starting time. The open shop problem can be solved in linear time, while the flow shop problem is NP-hard, but admits a pseudopolynomial algorithm and a fully polynomial approximation scheme. The flow shop problem with no-wait in process constraint admits a polynomial time approximation scheme.

Not only the complexity gap between the first two models is interesting to notice, but also the fact the two-machine open shop problem with a single fixed non-availability interval is NP-hard, while in our settings it is polynomially solvable.As topics for further research we mention the clarification of the complexity status of the flow shop no-wait problem subject to preventive

maintenance and study of approximability issues of its extended versions, e.g., with more machines or more maintenance intervals. Similar problems for other shop scheduling problems also are worth of investigating.

The obtained results should be seen as the first steps towards more complex models capable of handling more general types of machine environment and multiple machine maintenance intervals. The methods described in this chapter may serve as subroutines for possible constructive heuristics and metaheuristic algorithms for those generalized problems.

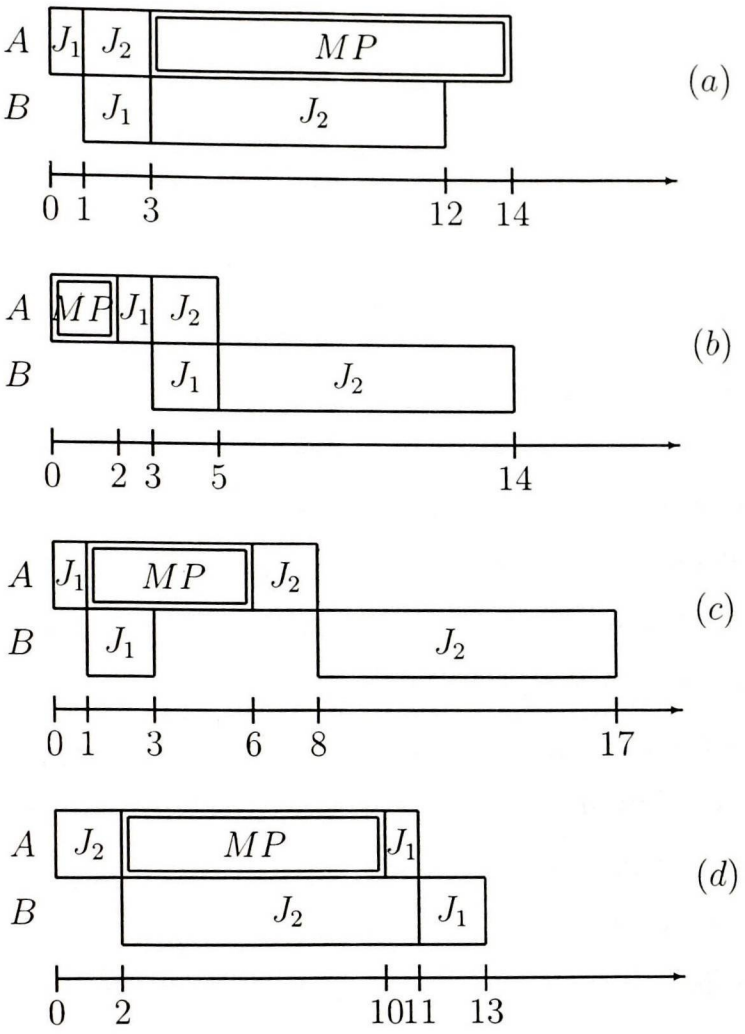

Figure 6.6: Schedules with the optimal Gilmore-Gomory permutation (J_1, J_2) with the MP placed (a) after the last job, (b) before the first job and (c) between the jobs; (d) optimal schedule with the sequence (J_2, J_1) and the MP between the jobs

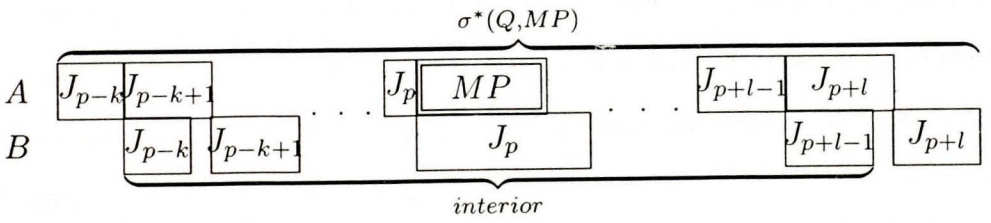

Figure 6.7: Block $\sigma^*(Q, MP)$

Chapter 7

Summary

The main topic of this research is the shop scheduling problems in which the processing machines may become non-available during the scheduling period. A number of such scheduling problems are considered. For some problems their complexity statuses are established and pseudopolynomial time algorithms are presented. For one problem a polynomial algorithm is developed which solves the problem optimally. For problems which are proved to be NP-hard either a polynomial time approximation algorithms with a finite worst-case ratio bound or (fully-)polynomial time approximations schemes are developed. Below we outline the obtained results as well as some possible directions for the future research. For all problems considered the objective is to minimise the makespan.

- For problem $F2\,|h(q,0),Re|\,C_{\max}$ we have developed a pseudopolynomial dynamic programming algorithm and showed that this algorithm can be extended for more general problem $F2\,|h(q_A,q_B),Re|\,C_{\max}$, see Section 3.2. This result improves the known pseudopolynomial dynamic programming algorithm for this problem with a single non-availability interval due to Lee [89].

- For problems $F2\,|h(1,0),Re|\,C_{\max}$ and $F2\,|h(0,1),Re|\,C_{\max}$ the correspondent pseudopolynomial dynamic programming algorithm by Lee

[89] is transformed into an FPTAS using the rounding technique. The same result was obtained independently by Ng and Kovalyov [107]. This result improves the known 4/3-approximation algorithms for the cases of one hole on one of the machines due to Lee [90] and Cheng and Wang [32]. The problem with several non-availability intervals on the second machine cannot be approximated within any finite factor, unless P=NP.

- Since the presented FPTAS has a high running time we propose a fast 3/2-approximation algorithm for the case of several non-availability intervals on the first machine, see Section 3.4. This algorithm improves and simplifies the known 3/2-approximation algorithm due to Lee [90] for the case of one hole on the first machine and it handles a more common case of several holes on the first machine. Cheng and Wang [32] have presented a 4/3-approximation algorithm for problem $F2\,|h(1,0), Re|\,C_{\max}$ but it cannot be used for the problem with several holes on the first machine.

- For problems $F2\,|h(0,1), S-Re|\,C_{\max}$ and $F2\,|h(1,0), S-Re|\,C_{\max}$ we present a polynomial time approximation scheme, see Section 3.5. This result improves the known 2-approximation algorithm for the problem with one hole on the first machine and 3/2-approximation algorithm for the problem with one hole on the second machine, see Lee [90]. It remains an open question whether the problem with a single hole under the semi-resumable scenario admits an FPTAS.

- For scheduling problems $F2\,|h(0,1), no-wait, Re|\,C_{\max}$ and $F2\,|h(1,0), no-wait, Re|\,C_{\max}$, $F2\,|h(0,1), no-wait, S-Re|\,C_{\max}$ and $F2\,|h(1,0), no-wait, S-Re|\,C_{\max}$ we establish their NP-hardness, see Section 4.2. Previously, this complexity result was established only for problems $F2\,|h(0,1), no-wait, N-Re|\,C_{\max}$ and

$F2\,|h(1,0), no{-}wait, N{-}Re|\,C_{\max}$, see [38, 39].

- We present 3/2-approximation algorithm for the two-machine flow shop problem with no-wait in process under any scenario, see Section 4.3. Only the non-resumable scenario has been considered for this problem in the literature. Thus, we obtained an improvement upon the best previously known 5/3-approximation algorithm for the non-resumable scenario due to [32]. Recently Cheng and Liu [30] present a polynomial time approximation algorithm for problems $F2\,|h(0,1), no{-}wait, N{-}Re|\,C_{\max}$ and $F2\,|h(1,0), no{-}wait, N{-}Re|\,C_{\max}$. It remains an open question whether this problem with a single non-availability interval under any scenario admits an FPTAS.

- In Section 4.4 we consider problems $F2\,|h(0,1), no{-}wait, Re|\,C_{\max}$ and $F2\,|h(1,0), no{-}wait, Re|\,C_{\max}$ which have not been considered in the literature before. We propose a 4/3-approximation algorithm for each of these problems.

- For the two-machine open shop problem we have developed a PTAS for the cases with either several holes on one of the machines or with a single hole on each of the machines under the resumable scenario. It significantly improves the known 4/3-approximation algorithm by Breit et al. [18] which can deal only with problems $O2\,|h(0,1), Re|\,C_{\max}$ and $O2\,|h(1,0), Re|\,C_{\max}$. Further, it is worth studying whether this problem with a single hole on one of the machines admits an FPTAS. Also, it is interesting to study the open shop problem with more than two processing machines and a non-availability interval on one of the machines.

- We try to initiate the study on scheduling problems in which machines have to be maintained during the planning period where the lengths of

each maintenance period is not fixed and depends on the start time of this maintenance. We concentrate on two-machines scheduling problems in which each machine has to be maintained exactly once during the scheduling period. For problem $O2\,|m(1,1)|\,C_{\max}$ we present a polynomial-time optimal algorithm.

- For problem $F2\,|m(1,1)|\,C_{\max}$ we prove that this problem becomes NP-hard even if the length of the maintenance interval depends linearly on its starting time. We also give a pseudopolynomial dynamic programming algorithm and two approximation algorithms, including a fully polynomial approximation scheme.

- A polynomial-time approximation scheme is designed for problems $F2\,|m(1,0), no{-}wait|\,C_{\max}$ and $F2\,|m(0,1), no{-}wait|\,C_{\max}$.

Index

Bibliography

[1] J.O. Achugbue and F.Y. Chin, Scheduling the open shop to minimise mean flow time, *SIAM Journal on Discrete Mathematics*, 11 (1982), 709-720.

[2] I. Adiri, J. Bruno, E. Frostig and A.H.G. Rinnooy Kan, Single machine flow-time scheduling with a single breakdown, *Acta Informatica*, 26 (1989), 679-696.

[3] A. Agnetis, P.B. Mirchandani, D. Pacciarelli and A. Pacifici, Scheduling problems with two competing agents, *Operations Research*, 52 (2004), 229–242.

[4] J.M. van den Akker, J.A. Hoogeveen and G.J. Woeginger, The two-machine open shop problem: To fit or not to fit, that is the question, *Operations Research Letters*, 31 (2003), 219–224.

[5] V.A. Aksjonov, A polynomial-time algorithm of approximate solution of a scheduling problem, *Upravlyaemye Sistemy*, 28 (1988), 8-11, (in Russian).

[6] S. Anily, C.A. Glass and R. Hassin, The scheduling of maintenance service, *Discrete Applied Mathematics*, 82 (1998), 27-42.

[7] S. Anily, C.A. Glass and R. Hassin, Scheduling maintenance services to three machines, *Annals of Operations Research*, 86 (1999), 375-391.

[8] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, Proof verification and hardness of approximation problems, *Journal of the ACM*, 45 (1998), 501–555.

[9] K.R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.

[10] A. Bar-Noy, R. Bhatia, J.S. Naor and B. Schiber, Minimising service and operation cost of periodic scheduling, *Mathematics of Operations Research*, 27 (2002), 518-544.

[11] I. Bárány and T. Fiala, Nearly optimum solution of multimachine scheduling problems, *Szigma*, 15 (1982), 177-191, (in Hungarian).

[12] G. D. Birkhoff, Tres observaciones sobre el algebra lineal, Universidad Nacional de Tucuman Revista, Serie A, 5 (1946), 147–151.

[13] J. Blazewicz, J. Breit, P. Formanowicz, W. Kubiak and G. Schmidt, Heuristic algorithms for the two-machine flowshop with limited machine availability, *Omega*, 29 (2001), 599-608.

[14] S.A. Borodich, On intractability of problems of minimisation of the number of tardy jobs in a two-machine open shop. In: *Complexity and Solution Methods of Optimization Problems*, Minsk (1984), 4-8 (in Russian).

[15] S.A. Borodich, On a problem of minimising the maximum lateness in a two-machine open shop, *Izvestia Akademii nauk BSSR*, Seria Fiziko-Matematicheskikh Nauk, 5 (1985), 109 (in Russian).

[16] J.-L. Bouqard, J.-C. Billaut, M.A. Kubzin, V.A. Strusevich, Two-machine flow shop scheduling problem with no-wait jobs, *Operations Research Letters*, 33 (2005), 255-262.

[17] J. Breit, Heuristische Ablaufplanungsverfahren für Flowshops und Openshops mit beschränkt verfügbaren Prozessoren, Ph.D. Thesis, University of Saarland, Saarbrücken, 2000.

[18] J. Breit, G. Schmidt and V.A. Strusevich, Two-machine open shop scheduling with an availability constraint, *Operations Research Letters*, 29 (2001), 65-77.

[19] J. Breit, G. Schmidt and V.A. Strusevich, Non-preemptive two-machine open shop scheduling with non-availability constraints, *Mathematical Methods of Operations Research*, 57 (2003), 217-234.

[20] J. Breit, An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint, *Information Processing Letters*, 90 (2004), 273 - 278.

[21] R.E. Burkard, V.G. Daĭneko, R. van Dal, J.A.A. van der Veen and G.J. Woeginger, Well-solvable cases of the travelling salesman problem: A survey, *SIAM Reviews*, 40 (1998), 496-546.

[22] B. Carr and S. Vempala, Towards a 4/3-approximation for the asymmetric travelling salesman problem, In: *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, (2000), 116-125.

[23] S. Y. Chang, H.-C. Hwang, The worst-case analysis of the MULTIFIT algorithm for scheduling nonsimultaneous parallel machines, *Discrete Applied Mathematics*, 92 (1999), 135-147.

[24] B. Chen, C.A. Glass, C.N. Potts and V.A. Strusevich, A new heuristic for three–machine flow shop scheduling, *Operations Research*, 44 (1996), 891–898.

[25] B. Chen, C.N. Potts and G.J. Woeginger, A review of machine scheduling: complexity, algorithms and approximability. In: D.-Z. Du and P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer, Dordrecht, 1998, pp. 21-169.

[26] B. Chen and V.A. Strusevich, Approximation algorithms for three machine open shop scheduling, *ORSA Journal on Computing*, 5 (1993), 321-326.

[27] B. Chen and V.A. Strusevich, Worst-case analysis of heuristics for open shops with parallel machines, *European Journal of Operational Research*, 70 (1993), 379-590.

[28] B. Chen and W. Yu, How good is a dense shop schedule?, *Acta Mathematicae Applicatae Sinica,* 17 (2001), 121-128.

[29] T.C.E. Cheng, Q. Ding and B.M.T. Lin, A concise survey of scheduling with time-dependent processing times, *European Journal of Operational Research,* 152 (2004), 1-13.

[30] T.C.E. Cheng, Z. Liu, Approximability of two-machine no-wait flowshop scheduling with availability constraints, *Operations Research Letters,* 31 (2003), 319-322.

[31] T.C.E. Cheng, G. Wang, Two-machine flowshop scheduling with consecutive availability constraints, *Information Processing Letters,* 71 (1999), 49-54.

[32] T.C.E. Cheng and G. Wang, An improved heuristic for two-machine flowshop scheduling with an availability constraint, *Operations Research Letters* 26 (2000), 223-229.

[33] Y. Cho and S. Sahni, Preemptive scheduling of independent jobs with release and due dates times on open, flow and job shop, *Operations Research,* 29 (1981), 511-522.

[34] E.G. Coffman Jr. (Ed.), *Scheduling in computer and job shop systems,* J. Willey, New York, (1976).

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

[36] S.A. Cook, The complexity of theorem proving procedures, in: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, ACM-Press, New-York, (1971), 151-158.

[37] J. Du and J.Y.-T. Leung, minimising mean flow time in two-machine open shops and flow shops, *Journal of Algorithms,* 14 (1993), 341-364.

[38] M.L. Espinouse, P. Formanowicz, B. Penz, minimising the makespan in the two-machine no-wait flow-shop with limited machine availability, *Computers & Industrial Engineering,* 37 (1999), 497-500.

[39] M.L. Espinouse, P. Formanowicz, B. Penz, Complexity results and approximation algorithms for the two machine no-wait flow-shop with limited machine availability, *Journal of the Operational Research Society,* 52 (2001), 116-121.

[40] U. Feige and C. Scheideler, Improved bounds for acyclic job shop scheduling, *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC'98)*, 624-633.

[41] S. French, Sequencing and scheduling: An introduction to the mathematics of the job-shop, Horwood, Chichester, (1982).

[42] A. Frieze, G. Galbiati and F. Maffioli, On the worst case performance of some algorithms for the symmetric travelling salesman problem, *Networks*, 12 (1982), 23-39.

[43] H.N. Gabow and O. Kariv, Algorithms for edge coloring bipartite graphs and multigraphs, *SIAM Journal on Computing*, 4 (1982), 117-129.

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

[45] M.R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, (1979).

[46] P.C. Gilmore, R.E. Gomory, Sequencing a one-state variable machine: a solvable case of the traveling salesman problem, *Operations Research*, 12 (1964), 655-679.

[47] P.C. Gilmore, E.L. Lawler, and D.B. Shmoys, Well-solved special cases, in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.) The Travelling Salesman Problem, John Wiley, Chichester, 1985, 87-143.

[48] A.A. Gladky, A two-machine preemptive openshop scheduling problem: an elementary proof of NP-completeness, *European Journal of Operational Research*, 103 (1997), 113-116.

[49] C.A. Glass, J.N.D. Gupta and C.N. Potts, Two-machine no-wait flow shop scheduling with missing operations, *Mathematics of Operations Research,* 24 (1999), 911-924.

[50] L.A. Golberg, M. Paterson, A. Srinivasan and E. Sweedyk, Better approximation guarantees for job-shop scheduling, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, 599-608.

[51] T. Gonzalez, A note on open shop preemptive schedules, Unit execution time shop problems, *IEEE Transactions on Computing,* 28 (1979), 782-786.

[52] T. Gonzalez and S. Sahni, Open shop scheduling to minimise finish time, *Journal of the Association of Computing Machinary,* 12 (1976), 665-679.

[53] T. Gonzalez and S. Sahni, Flowshop and jobshop schedules: complexity and approximation, *Operations Research,* 26 (1978), 36–52.

[54] M. Gopalakrishnan, S.L. Ahire and D.M. Miller, Maximizing the effectiveness of a preventive maintenance system: an adaptive modeling approach, *Management Science*, 43 (1997), 827-840.

[55] S.K. Goyal, A note on paper: On the flowshop sequencing problem with no-wait in process, *Operational Research Quarterly*, 24 (1973), 130-133.

[56] S.K. Goyal and C. Sriskandarajah, No-wait scheduling: computational complexity and approximation algorithms, *Opsearch*, 25 (1988), 220-244.

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

[58] G.H. Graves, and C.-Y. Lee, Scheduling maintenance and semiresumable jobs on a single machine, *Naval Research Logistics*, 46 (1999), 845-863.

[59] A. Grigoriev, J. van de Klundert and F.C.R. Spieksma, Modelling and solving periodic maintenance problem, Working paper, Maastricht University, Maastricht, The Netherlands, (2002).

[60] L.A. Hall, A polynomial time approximation scheme for a constrained flow-shop scheduling problem, *Mathematics of Operations Research*, 19 (1994), 68-85.

[61] L.A. Hall, Approximability of flow shop scheduling, *Mathematical Programming*, 82 (1998), 175-190.

[62] N.G. Hall and C. Sriskandarajah, A survey of machine scheduling problems with blocking and no-wait in process, *Operations Research*, 44 (1996), 510-525.

[63] Y. He, Parametric LPT-bound on parallel machine scheduling with nonsimultaneous machine available time, *Asia-Pacific Journal of Operational Research*, 15 (1998), 29-36.

[64] J.A. Hoogeveen and T. Kawaguchi, Minimising total completion time in a two-machine flowshop: analysis of special cases, *Mathematics of Operations Research*, 24 (1999), 887-910.

[65] J.A. Hoogeveen, P. Schuurman and G.J. Woeginger, Non-approximability results for scheduling problems with minsum criteria, Technical Report Woe-15, Department of Mathematics, TU Graz, Graz, Austria (1997).

[66] H.-C. Hwang and S.Y. Chang, Parallel machines scheduling with machines shutdowns, *Computers and Mathematics with Applications*, 36 (1998), 21-31.

[67] J.R. Jackson, An extension of Johnson's results on job lot scheduling, *Naval Research Logistics Quarterly*, 3 (1956), 201-203.

[68] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly,* 1 (1954), 61-68.

[69] S.N. Kabadi, M.F. Baki, Gilmore-Gomory type travelling salesman problems, *Computers and Operations Research*, 26 (1999), 329-351.

[70] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (eds.), Complexity of Computer Communications, New York: Plenum Press, (1972), 85-103.

[71] M. Kaspi and B. Montreuil, On the scheduling of identical parallel processes with arbitrary initial processor available time, Research Report 88-12, School of Industrial Engineering, Purdue University, (1988).

[72] H. Kellerer, Algorithms for multiprocessor scheduling with machine release time, *IIE Transactions*, 30 (1998), 991-999.

[73] A. Kononov, S. Sevastianov and I. Tschernykh, Polynomially solvable classes of the open shop problem on the base of different machine loads, submitted to *Annals of Operations Research,* 92 (1999), 211-239.

[74] M.Y. Kovalyov and F. Werner, A polynomial approximation scheme for problem $F2|r_j|C_{\max}$, *Operations Research Letters*, 20 (1997), 75-79.

[75] W. Kubiak, C. Sriskandarajah and K. Zaras, A note on the complexity of openshop scheduling problems, *INFOR*, 29 (1991), 284-293.

[76] W. Kubiak, J. Błażewicz, P. Formanowicz, J. Breit and G. Schmidt, Two-machine flow shops with limited machine availability, *European Journal of Operational Research*, 136 (2002), 528-540.

[77] M.A. Kubzin, C.N. Potts and V.A. Strusevich, Approximation schemes for the flow shop scheduling problems with non-availability constraints, *Book of abstracts, MAPSP'03,* (2003), 76-77.

[78] M.A. Kubzin, V.A. Strusevich, J. Breit and G. Schmidt, Polynomial-time approximation schemes for the open shop scheduling problem with non-availability constraints, *Paper No. 02/IM/100, CMS Press, University of Greenwich*, (2002); to appear in *Naval Research Logistics*.

[79] M.A. Kubzin, V.A. Strusevich, J. Breit and G. Schmidt, The open shop scheduling problem with non-availability constraints: polynomial-time approximation schemes, *Book of abstracts, MAPSP'03*, (2003), 141-142.

[80] M.A. Kubzin and V.A. Strusevich, An approximation algorithm for the two-machine flow shop no-wait scheduling problem with a non-availability interval, *Book of abstracts, CO'02*, (2002), 73.

[81] M.A. Kubzin and V.A. Strusevich, Two-machine flow shop no-wait scheduling with a nonavailability interval, *Naval Research Logistics*, 51 (2004), 613-631.

[82] M.A. Kubzin and V.A. Strusevich, Planning machine maintenance in two-machine shop scheduling, *Paper No. 04/IM/111, CMS Press, University of Greenwich*, (2004).

[83] M.A. Kubzin and V.A. Strusevich, Two-machine flow shop no-wait scheduling with machine maintenance, to appear in *4OR*.

[84] E.L. Lawler and J. Labetoulle, On preemptive scheduling of unrelated parallel processors by linear programming, *Journal of the ACM*, 25 (1978), 612-619.

[85] E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, minimising maximum lateness in a two-machine open shop, *Mathematics of Operations Research*, 6 (1981), 153-158. Erratum: *Mathematics of Operations Research*, 7 (1982), 635.

[86] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, "Sequencing and scheduling: algorithms and complexity," in Handbooks in *Operations*

Research and Management Science, vol. 4, Logistics of Production and Inventory, S.C. Graves, A.H.G. Rinnooy Kan, P.H. Zipkin (Editors), North–Holland, Amsterdam, (1993), 455-522.

[87] C.-Y. Lee, Parallel machines scheduling with non-simultaneous machine available time, *Discrete Applied Mathematics*, 30 (1991), 53-61.

[88] C.-Y. Lee, Machine scheduling with an availability constraint, *Journal of Global Optimization*, 9 (1996), 395-416.

[89] C.-Y. Lee, minimising the makespan in the two-machine flowshop scheduling problem with an availability constraint, *Operations Research Letters*, 20 (1997), 129-139.

[90] C.-Y. Lee, Two-machine flowshop scheduling with availability constraints, *European Journal of Operational Research,* 114 (1999), 420-429.

[91] C.-Y. Lee, Machine scheduling with availability constraints, in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, J. Leung (Editor), Chapman & Hall/CRC, London, (2004), pp. 22-1 – 22-13.

[92] C.-Y. Lee, and Z.-L. Chen, Scheduling jobs and maintenance activities on parallel machines, *Naval Research Logistics*, 47 (2000), 145-165.

[93] C.-Y. Lee and S.D. Liman, Single machine flow-time scheduling with scheduled maintance, *Acta informatica* 29 (1992), 375-382.

[94] C.-Y. Lee and S.D. Liman, Capacitated two-parallel machine scheduling to minimise sum of job completion times, *Discrete Applied Mathematics,* 41 (1993), 211-222.

[95] C.-Y. Lee and C.-S. Lin, Single-machine scheduling with maintenance and repair rate-modifying activities, *European Journal of Operational Research*, 135 (2001), 493-513..

[96] J.K. Lenstra, Sequencing by enumerative methods, Mathematical Centre Tract 69, Amsterdam, (1977).

[97] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, *Annals of Operations Research*, 1 (1977), 343-362.

[98] J.Y.-T. Leung and M. Pinedo, A note on scheduling parallel machines subject to breakdown and repair, *Naval Research Logistic*, 51 (2004), 60-71.

[99] C.Y. Liu and R.L. Bulfin, On the complexity of preemptive open-shop scheduling problems, *Operations Research Letters*, 4 (1985), 71-74.

[100] T. Lorigeon, J.-C. Billaut and J.-L. Bouquard, A dynamic programming algorithm for scheduling jobs in a two-machine open shop with an availability constraint, *Journal of the Operational Research Society*, 53 (2002), 1239-1246.

[101] L. Lu and M.E. Posner, An NP-hard open shop scheduling problem with polynomial average time complexity, *Mathematics of Operations Research*, 18 (1993), 12-38.

[102] C.L. Monma and A.H.G. Rinnooy Kan, A concise survey of efficiently solvable special cases of the permutation flow-shop problem, *RAIRO Recherche opérationnelle*, 17 (1983), 105-119.

[103] J.M. Moore, An n job one machine sequencing algorithm for minimising the number of late jobs, *Management science*, 15 (1968), 102-109.

[104] G. Mosheiov, Minimising the sum of job completion times on capacitated parallel machines, *Mathematics of Computer Modelling*, 20 (1994), 91-99.

[105] J. von Neumann, A certain zero-sum two-person game equivalent to an optimal assignment problem, *Contributions to the Theory of Games*, 28 (1953), 5-12.

[106] Y.D. Neumytov and S.V. Sevastianov, An approximation algorithm with an exact bound for the three–machine problem with the opposite routes, *Upravlyaemye Sistemy*, 31 (1993), 53–65 (in Russian).

[107] C.T. Ng and M.Y. Kovalyov, An FPTAS for scheduling a two-machine flow-shop with one unavailability interval, *Naval Research Logistics*, 51 (2004), 307-315.

[108] E. Nowicki and C. Smutnicki, Worst-case analysis of an approximation algorithm for flow-shop scheduling, *Operations Research Letters*, 8 (1989), 171-177.

[109] E. Nowicki and C. Smutnicki, Worst-case analysis of Dannenbring's algorithm for flow-shop scheduling, *Operations Research Letters*, 10 (1991), 473-480.

[110] E. Nowicki and C. Smutnicki, New results in the worst-case analysis for flow-shop scheduling, *Discrete Applied Mathematics*, 46 (1993), 21-41.

[111] E. Nowicki and C. Smutnicki, A note on worst-case analysis of an approximation algorithms for a scheduling problem, *European Journal of Operational Research*, 74 (1994), 128-134.

[112] D. Nyman, and J. Levitt, *Maintenance Planning, Scheduling and Coordination*, Industrial Press, (2002).

[113] D. Palmer, *Maintenance Planning and Scheduling Handbook*, McGraw Hill, (1999).

[114] C.H. Papadimitriou and P.C. Kannelakis, Flow shop scheduling with limited temporary storage, *Journal of the ACM*, 27 (1980), 533-549.

[115] C.H. Papadimitriou and M. Yannakakis, Optimisation, approximation and complexity classes, *Journal of Computer and System Sciences*, 43 (1991), 425-440.

[116] J. Piehler, Ein Beitrag zum Reinhenfolgeproblem, *Unternehmensforschung*, 4 (1960), 138-142.

[117] M. Pinedo and L. Schrage, Stochastic shop scheduling: a survey, *Deterministic and Stochastic Scheduling*, M.A.H. Dempster et al. (eds.), Riedel, Dordrecht, (1982), 181–196.

[118] X. Qi, T. Chen and F. Tu, Scheduling the maintenance on a single machine, *Journal of Operatinal Research Society*, 50 (1999), 1071-1078.

[119] A.H.G. Rinnooy Kan, Machine scheduling problems: classification, complexity and computations, Martinus Nijhoff, Hague, 1976.

[120] G. Rote and G.J. Woeginger, Time complexity and linear-time approximation of the ancient two machine flow shop, *Journal of Scheduling*, 1 (1998), 149-155.

[121] H. Röck and G. Schmidt, Machine aggregation heuristics in shop-scheduling, *Methods of Operations Research*, 45 (1983), 303-314.

[122] H. Röck, The three-machine no-wait flow shop is NP-complete, *Journal of the ACM*, 31 (1984), 336-345.

[123] H. Röck, Some new results in flow shop scheduling, *ZOR - Mathematical Methods of Operations Research*, 28 (1984), 1-16.

[124] C.N. Potts, Analysis of heuristics for two-machine flow-shop sequencing subject to release dates, *Mathematics of Operations Research*, 10 (1985), 576-584.

[125] C.N. Potts, D.B. Shmoys and D.P. Williamson, Permutation vs. non-permutation flow shop schedules, *Operations Research Letters,* 10 (1991), 281-284.

[126] S.S. Reddi, C.V. Ramamoorthy, On the flow shop sequencing problem with no-wait in process, *Operational Research Quarterly,* 23 (1972), 323-331.

[127] C. Sadfi, B. Penz, C. Rapine, J. Błażevich and P. Formanowicz, An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints, *European Journal of Operational Research*, 161 (2005), 3-10.

[128] C. Sadfi, B. Penz and C. Rapine, A dynamic programming algorithm for the single machine total completion time scheduling problem with availability con-

straints, *8th International Workshop on Project Management and Scheduling - PMS,* Valencia, Spain, (2002).

[129] S. Sahni and Y. Cho, Complexity of scheduling shops with no wait in process, *Mathematics of Operations Research*, 4 (1979), 448-457.

[130] E. Sanlaville and G. Schmidt, Machine scheduling with availability constraints, *Acta Informatica*, 35 (1998), 795-811.

[131] G. Schmidt, Scheduling with limited machine availability, *European Journal of Operational Research*, 121 (2000), 1-15.

[132] S.V. Sevatianov, Efficient scheduling in the open shop systems, *Siberian Journal of Operations Research*, 1 (1994), 20-42 (in Russian).

[133] S.V. Sevast'janov, On some geometric methods in scheduling theory, *Discrete Applied Mathematics,* 55 (1994), 59–82.

[134] S.V. Sevastianov and G.J. Woeginger, Makespan minimisation in open shops: a polynomial time approximation scheme, *Mathematical Programming*, 82 (1998), 191-198.

[135] N.V. Shakhlevich and V.A. Strusevich, Two machine open shop scheduling problem to minimise an arbitrary machine usage penalty function, *European Journal of Operational Research*, 70 (1993), 391-404.

[136] D.B. Shmoys, C. Stein and J. Wein, Improved approximation algorithms for shop scheduling problems, *SIAM Journal on Computing*, 23 (1994), 617-632.

[137] W.E. Smith, Various optimizers for single state production, *Naval Research Logistics Quarterly,* 3 (1956), 69-66.

[138] C. Sriskandarajah and E. Wagneur, On the complexity of preemptive open shop scheduling problems, Cahier du GERAD, G-90-36, Ecole des Hautes Etudes Commerciales, Montreal, Canada (1990).

[139] V.A. Strusevich, A greedy open shop heuristic with job priorities, *Annals of Operations Research*, 83 (1998), 253-270.

[140] M. Sviridenko and G.J. Woeginger, Makespan Minimization in No-Wait Flow Shops: A Polynomial Time Approximation Scheme, *SIAM Journal on Discrete Mathematics,* 16 (2003), 313-322.

[141] W. Szwarc, The flow-shop problem with mean completion time criterion, *IIE Transactions*, 15 (1983), 172-176.

[142] V.S. Tanaev, Y.N. Sotskov and V.A. Strusevich, *Scheduling Theory. Multi-Stage Systems*, Kluwer, Dordrecht, 1994.

[143] G. Vairaktarakis and S. Sahni, Dual criteria preemptive open–shop problems with minimum makespan, *Naval Research of Logistics*, 42 (1995), 103-121.

[144] P.M. Vaidya, Speeding up linear programming using fast matrix multiplication, In: *Proceedings of IEEE 30th Annual Symposium on Foundations of Computer Science*, (1989), 332–337.

[145] J.A.A. van der Veen and R. van Dal, Solvable cases of the no-wait flow-shop scheduling problem, *Journal of the Operational Research Society,* 42 (1991), 971-980.

[146] S.L. van de Velde, minimising the sum of the job completion times in the two-machine flow shop by Lagrangian relaxation, *Annals of Operations Research*, 26 (1990), 257-268.

[147] G. Wang, T.C.E. Cheng, Heuristics for two-machine no-wait flowshop scheduling with an availability constraint, *Information Processing Letters* 80 (2001), 305-309.

[148] J.M. Wein, *Algorithms for Scheduling and Network Problems*, PhD. Thesis, Cambridge, USA (1991).

[149] D. de Werra, On some combinatorial problems arising in scheduling, *CORS Journal*, 8 (1970), 165-175.

[150] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevast'janov and D.B. Shmoys, Short shop schedules, *Operations Research,* 45 (1997), 288–294.

[151] D.A. Wismer, Solution of the flow shop scheduling problem with no intermediate queues, *Operations Research*, 20 (1972), 689-697.