SHOP SCHEDULING IN MANUFACTURING SYSTEMS: ALGORITHMS AND COMPLEXITY

By

ZHIHUI XUE, B.ENG., M.ENG.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

SCHEDULING MANUFACTURING SYSTEMS

Doctor of Philosophy (2004) McMaster University

(Management Science/Systems) Hamilton, Ontario

TITLE: Shop Scheduling in Manufacturing Systems: Algorithms and Complexity

AUTHOR: Zhihui Xue, B.Eng. (Nanchang University, Jiangxi, China), M.Eng. (Tsinghua University, Beijing, China)

SUPERVISOR: Professor George Steiner

NUMBER OF PAGES: ix, 91

Abstract

This thesis describes efficient algorithms and complexity results for some machine scheduling and related problems, which are encountered in automated manufacturing systems.

We introduce a new class of robotic-cell scheduling models. The novel aspect is that parts need to reenter machines several times before they are finished. The problem is to find the sequence of robot move cycles and the part processing sequence that jointly minimize the cycle time or the makespan. We show that the problems are computationally intractable with three machines and present polynomial solutions for a variety of two-machine configurations.

We then consider the problem of scheduling multi-component parts in a two-machine robotic cell, where each part is composed of K identical components to be processed together on the first machine, then processed on the second machine individually. We study the cycle time and makespan minimization problems, and show that both are polynomially solvable.

We investigate the problem of minimizing cycle time in a two-machine job shop, where each job has at most three operations. We reduce the problem to a two-machine reentrant flow shop problem. By extending previous results on the reentrant flow shop problem, we propose a new pseudo-polynomial algorithm, as well as a fully polynomial-time approximation scheme for certain special cases of the job shop problem. We also describe a 4/3-approximation algorithm for the general problem, and identify several well-solvable cases.

Finally, we study special cases of the traveling salesman problem on permuted Monge matrices, which arose from robotic-cell scheduling problems. By using the theory of subtour patching, we reduce the problems to finding a minimum-b-weight spanning tree in the patching graph. In general, this problem is $\mathcal{NP}$-hard. We show, however, that newly defined special properties of the distance matrix allow us to find in polynomial time a minimum-b-weight spanning tree, and thus an optimal tour, for these new classes.

Preface

This thesis investigates the problems of scheduling automated production processes and related problems. The study was sparked by real-world applications in a hi-tech company which designs and manufactures state-of-the-art industrial robots (motion control devices) for the biotechnology and pharmaceutical segments of the global Life Sciences market. For detailed motivations, the reader is referred to the Introduction in Chapters 2 and 3.

A major thrust of research on production scheduling and sequencing is the design, analysis, implementation, and experimental evaluation of algorithms to solve critical problems. In this study, we focus on the design and analysis aspects of algorithms. The philosophy we take is to introduce a scheduling model, such as reentrant robotic cell, and then discuss algorithmic issues for such model and its associated problems. Due to the fact that we are devoted to the theoretic (complexity) analysis of algorithms, the reader might notice that the scheduling models dealt with in this thesis are "fundamental"—in the sense they have been abstracted and simplified a lot from reality.

The thesis consists in part of previously prepared materials. Parts of Chapters 2 and 4 are based on two research papers—"Scheduling in reentrant robotic cells: Algorithms and complexity" and "On minimizing cycle time in a two-machine job shop"—coauthored with George Steiner. For these two papers and their results, I am the primary contributor. Chapter 5 is adapted from the research paper entitled "New solvable cases of the traveling salesman problem on permuted Monge matrices," which is a joint work with Vladimir G. Deĭneko and George Steiner. The paper came into being from a special case of the traveling salesman problem (TSP) I had studied and solved in scheduling multi-component parts in a robotic cell (see Chapter 3 for details). Using the methodology from Deĭneko, a unified approach for the TSP on more general distance matrices was obtained and presented. All of the research documented in this thesis has been carried out during my doctoral study.

Acknowledgements

A doctoral study is not only about solving problems and then writing thesis, but also about gathering new insight. During this learning process, I owed a lot to my mentor and advisor, George Steiner, who introduced me to the field of scheduling. Since the beginning, he has been actively involved in this research. The many discussions of the fundamentals of the subject have always been very stimulating to me. I have enjoyed working with him and learned from him because of his enthusiasm and his inspiring ideas. I hereby express my sincere thanks to him for his constant support, guidance, and encouragement.

Apart from George Steiner, I would like to show my appreciation to Prakash Abad and Stavros Kolliopoulos, for their careful reading of the thesis, their corrections, and their valuable suggestions to improve the style. Their classes (on Operations Management and Approximation Algorithms) have taught me many new ideas and approaches for which I am most thankful.

It has been a pleasure to work with Vladimir G. Deĭneko of the University of Warwick (Coventry, UK) on new solvable cases of the traveling salesman problem. Although our communications were via e-mail, I could always count on his clear, sound, and timely advice. Thanks Vladimir, this thesis has benefited from your intelligent advice. Chapter 4 would not have been written without the cooperation of Nicholas G. Hall. Many thanks for sending us your paper, which led us to study the cyclic job-shop problem in Chapter 4. Thanks to him also for being on my reading committee and his constructive comments on my work. I am also grateful to Joris van de Klundert for sending me his Ph.D. dissertation. Thanks to several anonymous referees for insightful comments on earlier versions of some parts of this thesis.

I would like to take this opportunity to thank Mahmut Parlar and Linda Kszyston for their help with all kinds of professional and administrative matters. In addition, I would especially like to express my gratitude to Paul Stephenson and Jinliang Cheng for their assistance in my initial settlement in Hamilton in a cold winter. I still remember the winter storm that I encountered the first time in my life when I arrived at the Pearson (Toronto) international airport in a chilly night.

The Ph.D. study could not have been finished without the whole-hearted support from my family. I am deeply indebted to my parents, whose love and encouragement were total and indispensable even though they were thousands of miles away.

Finally, I gratefully acknowledge financial support from McMaster Graduate Scholarships and Ontario Graduate Scholarships in Science and Technology. My thanks also go to Mathematics of Information Technology and Complex Systems (MITACS) for sponsoring my trips to its Annual General Meeting.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

"The journey of a thousand miles begins with one step."
—Lao Tzu

Scheduling is the science of allocating limited resources over time to complete a set of tasks. Typically, the result of allocation is specified in a schedule. For a given collection of tasks and resources, a schedule is *feasible* if it does not violate any accompanying constraints. Sometimes, finding a single feasible schedule is enough. However, under many circumstances, the goal is to find the best schedule from among all feasible schedules to achieve certain objective(s), such as the shortest schedule length or the maximum number of tasks completed before the due dates. Thus scheduling may be considered as an optimization process.

Relying largely on mathematical methodology, the formal research of scheduling began in the early 1950s. The pioneering work by Graham and Johnson has sparked extensive research in *scheduling theory.* Since then, scheduling theory has become one of the most active areas in operations research with a significant number of problems and models studied in the literature. The attraction of scheduling theory is not just for its deep mathematical basis but also for its impressive practical use. More and more of it is being applied to industrial processes and mission-critical computer systems, see Błażewicz et al. (2001) for an overview of scheduling in computer and manufacturing systems.

Traditionally, the research on scheduling is motivated by questions arising in production planning and manufacturing, which has led to the development of a theory of *machine scheduling.* In the last two decades, with many new types of manufacturing shops, such as flexible manufacturing systems (FMSs) and computer integrated manufacturing (CIM), coming into prominence, the research on machine scheduling has undergone a profound transition (Lee, Lei, and Pinedo 1997). Often, scheduling problems in modern manufacturing systems are more complex than classical scheduling problems because in most cases additional resources and constraints have to be satisfied. In some cases, even simultaneous decisions regarding connections between several limited resources are to be made. For instance, in many automated manufacturing environments, material handling is performed by computer-controlled robots, hoists, cranes or vehicles, rendering the performance of the systems highly dependent on the interaction between the machines and the material-handling devices, both are

the limited resources of interests (Ganesharajah, Hall, and Sriskandarajah 1998). A good schedule that synchronizes the activities of machine processing and material handling may not only increase throughput rate, but also reduce work-in-process and production costs. It is reported that there are more than 600 companies in the United States and Japan which develop and/or use advanced computer algorithms for the scheduling of material-handling devices (Kats and Levner 2002). Unfortunately, most of the classical scheduling models do not incorporate several of the distinct attributes of state-of-the-art manufacturing systems. Hence, from both the theoretical and the practical points of view, investigating new models to tackle these newly posed scheduling problems is of particular importance.

In this study we will look into several scheduling problems and related problems arising in connection with such manufacturing systems. The intention is to explore some typical problems and try to gain insights into more complicated real-world problems. In pursuing this line of research, our emphasis is on algorithmic results and complexity analysis, rather than empirical or simulation studies. Nevertheless, we attempt to give a detailed account of both the theory and applications. Models representing these problems will be formulated and solution methods will be investigated. For special situations, optimal polynomial-time algorithms will be developed. For some $\mathcal{NP}$-hard problems, much attention is paid to the exact complexity status and the design of efficient approximate solutions with guaranteed worst-case performance ratios.

This introductory chapter continues with an extensive explanation of machine scheduling and a classification of scheduling problems in Section 1.1. A new type of production method that is frequently encountered in modern manufacturing is described in Section 1.2. We end this chapter with an overview of the thesis in Section 1.3.

1.1 Preliminaries

The purpose of this section is to outline the classical scheduling models and relevant concepts that will be used later. We make every effort to adhere to traditional notation and standard terminology. The scheduling models we shall discuss are based on the *deterministic off-line machine scheduling* paradigm, where all data are assumed to be discrete (i.e., non-negative integers) and known with certainty in advance. Note that, in terms of this assumption, scheduling may be considered as a part of combinatorial optimization. Throughout this thesis we will make extensive use of concepts related to the design and analysis of algorithms. It is expected that the reader has some familiarity with those concepts which can, for instance, be found in the text of Cormen et al. (2001). For a rigorous and comprehensive discussion of computational complexity, the books by Garey and Johnson (1979) and Papadimitriou (1994) are two excellent sources.

1.1.1 Machine scheduling problems

In machine scheduling, the limited resources consist of one or more *machines* and tasks are modeled as *jobs* that can be executed by the machines. A task (job) first becomes available for processing at its ready time, and it must receive an amount of processing equal to its processing time. Typically, a problem in machine scheduling can be characterized by the types of machines and jobs in the system, by the scheduling constraints imposed on them, and by a desired optimality principle.

A characteristic of the machine environment is that a machine can handle at most one job at a time and that each job can be processed by only one machine at a time. In general, a machine can begin its next job immediately after the current job is completed, and there are no machine breakdowns at any moment in time. For all the scheduling problems we consider, it is assumed that *preemption* is not allowed during the processing of any operation, which means that the execution of a job on a machine will proceed without interruption once it starts.

A machine scheduling problem is in fact a *sequencing problem* if any schedule can be completely specified by the sequence in which jobs are performed. Being aware of this fact, we may use the name of schedule and the corresponding sequence interchangeably when there is no confusion.

In what follows, we describe classical scheduling models. We focus on the shop scheduling models that are frequently encountered in manufacturing shop floors. Several books in scheduling, e.g., Błażewicz et al. (2001), Brucker (2001), and Pinedo (2002) have contained more machine scheduling models, such as the models of single and parallel machines.

1.1.2 Shop scheduling models

In many manufacturing and production systems, jobs have to be processed by several machines in a given order. This multi-operation situation is often reflected in the so-called *shop scheduling* model, where a number of jobs is to be processed in a shop consisting of several machines. Usually, it is assumed that the machines have unlimited buffer space and a job can be stored in the buffer for an unlimited amount of time. If, however, the machines have limited buffer space, then *blocking* occurs when the buffer is full. In this case, the job at the upstream machine cannot be released into the buffer after completing its processing. It has to remain at the upstream machine, preventing a job in queue at that machine from beginning its processing.

To further define shop models, suppose that the order in which a job passes through the machines, also known as the *processing route*, is fixed for each job. For convenience, let us assume that any two consecutive operations of a job are to be processed on different machines. For otherwise, we can combine these two operations into a single operation. Two typical models are of interest in this context, namely,

flow shop and job shop.

The job shop model is one of the most general in scheduling theory. In a *job shop*, each job consists of a number of operations to be processed on all or some of the given machines, and each job has its own processing route to follow. Hence to construct a feasible schedule for a job shop, we have to determine, for each machine, an order in which the jobs are to be processed. Note that in a job shop, a job may visit a machine more than once, also a job may not visit a machine at all. The flow shop is a special case of the job shop. In a *flow shop*, each job requires processing on every machine only once and the processing route is identical for all jobs. In general, jobs are able to pass each other while they are waiting in queues at the machines for processing, provided that all jobs follow the same processing route. In other words, in a flow shop each machine may process the jobs in a different order. If, however, each machine processes the jobs in the same order—that is, no passing among jobs is allowed, the flow shop is referred to as a *permutation flow shop* and the schedule is said to be a *permutation* or *no-passing schedule.*

In the aforementioned shop models, there are no precedence relationships among jobs prescribing the order in which job processings must be carried out. While the machine sequence (i.e., the processing route) of all jobs is given, the scheduling problem is to find the best job processing sequence according to a desired optimality principle.

1.1.3 Scheduling classification

The seemingly infinite number of deterministic machine scheduling problems makes it clear that there is a need for classification. In general, deterministic machine scheduling problems may be represented using a *three-field notation* $\alpha|\beta|\gamma$ proposed by Graham et al. (1979). Simply, the three-field notation $\alpha|\beta|\gamma$ may be sketched as follows:

- The first field α indicates the machine environment. For instance, $\alpha = F$ or J denotes the flow shop or job shop model, respectively. The number of machines m is either part of the problem instance or equal to a fixed constant. In the latter case, the letter m or a positive integer is added after the machine environment, e.g., the two-machine job shop model is specified by $J2$.
- The second field β consists of the job characteristics, i.e., the processing restrictions and constraints. In contrast to the first field, this field can be empty, which implies the default of non-preemptive and independent jobs. Examples of possible entries in this field are $\beta = pmtn$, meaning that *preemption* is allowed (i.e., the processing of any operation may be interrupted and resumed at a later time), and $\beta = prec$, meaning that there are *precedence constraints* between the jobs (i.e., the processing of a job cannot start before the completion of another job).

- The third field γ specifies the optimality criterion or the objective. An optimality criterion assesses the relative merits or performances of competing feasible schedules. Examples of commonly used criteria include minimizing the *makespan* $C_{\max}$ (i.e., the maximum completion time of all jobs on all machines) and minimizing the total weighted completion time $\sum w_j C_j$ of all jobs.

Stating each variation of a scheduling problem by the three-field notation provides a quick reference point to facilitate comparisons between problems. For instance, the problem of minimizing makespan in an m-machine permutation flow shop is identified by the three-tuple $Fm|prmu|C_{\max}$, while the problem in a general m-machine flow shop is denoted by $Fm||C_{\max}$.

Above we have given a rough description of the classification scheme. For further details, the reader is referred to the survey paper by Lawler et al. (1993) and the book by Pinedo (2002).

1.2 Cyclic Production and Scheduling

Traditional production manufactures a given set of parts with the objective of minimizing the maximum completion time or *makespan*. In many modern, high-volume manufacturing environments, a so-called *cyclic production* is often adopted. In this method, rather than process a large batch of each part, a small set of parts is loaded into the system and processed repetitively. For example, consider the need to process 3,000 units of part A, 2,000 units of part B, and 4,000 units of part C in a day. A part mix ratio or *minimal part set* (MPS) is calculated as $\{3A, 2B, 4C\}$—namely, three of A, two of B, and four of C, which is the smallest set of parts in proportion to the day's production. Then the MPS is fed into the system and produced 1,000 times to fulfill the production target.

For an MPS, after the operations of each part are assigned to the machines, the processing order in which the operations are processed at a machine must be specified, and this order must be followed by each MPS. Moreover, the parts in an MPS are processed by the machines in a specified order. This order is the same for each MPS.

In this context, we develop the class of *cyclic schedules* that perform each required operation on an MPS exactly once. When such a schedule is formed, it will be identically repeated at regular intervals. To measure the performance of such a repetitive manufacturing system, where there is no need to track each individual order, one often-used objective is to minimize the *cycle time* of an MPS—the time between completions of successive MPSs, which is equivalent to maximizing the throughput rate over the long run. (Note that the cycle time objective is different from the makespan objective. The makespan objective minimizes the length of time from when production starts until it ends, i.e., the schedule length. Accordingly, algorithmic or complexity results for a problem with the makespan objective are not necessarily

valid for a problem with the cycle time objective, and vice versa. See Hall, Lee, and Posner (2002) and Chapters 2, 3 and 4 for details.) Due to its simplicity in management and control, cyclic production is suitable for producing large quantities of different parts which have small setup cost. Besides production systems, cyclic scheduling (i.e., constructing a recurrent schedule) arises in application areas like compiler design, digital signal processing, railway scheduling, and timetabling as well (see e.g., Stankovic et al. 1998 and Ernst et al. 2004).

1.3 Overview of the Thesis

The rest of the thesis consists of five chapters. For the sake of enhancing the readability of this thesis, Chapters 2 to 5, the main body of the thesis, are written so as to be coherent and fully self-contained in the sense that all formal concepts and arguments needed to analyze the problems in each chapter can be explained in detail. This implies that some of the chapters may contain redundant material. Each chapter is focused on a specific topic, and provides a complete and systematic account of the research. In each chapter, we provide an introduction that presents sufficiently relevant information to establish a problem context, and a concluding summary. Brief descriptions of the problems we study and highlights of our results follow. Note that in robotic-cell scheduling problems we prefer to use the term "part" instead of "job" to illustrate the concrete transporter activities. Actually, it has the same meaning as "job" in scheduling terminology.

Chapter 2 is devoted to the scheduling of m-machine reentrant robotic cells, where parts need to reenter machines several times before they are finished. The problem is to find the sequence of 1-unit robot move cycles and the part processing sequence which jointly minimize the cycle time or the makespan. When $m = 2$, we show that both the cycle time and the makespan minimization problems are polynomially solvable. When $m = 3$, we examine a special class of reentrant robotic cells with the cycle time objective. We show that in a three-machine loop-reentrant robotic cell, the part sequencing problem under three out of the four possible robot move cycles for producing one unit is strongly $\mathcal{NP}$-hard. The part sequencing problem under the remaining robot move cycle can be solved easily. Finally, we prove that the general problem, without restriction to any robot move cycle, is also intractable.

Chapter 3 is dedicated to scheduling multi-component parts in a two-machine manufacturing cell. The cell is served by a robot, which loads, unloads, and moves parts between machines. Each part is composed of K identical components to be processed together first on machine M_1, then processed on machine M_2 individually. The objective is to determine the best part processing sequence and the corresponding robot move cycles to produce the parts efficiently. We study the cycle time and makespan minimization problems. By showing that both problems can be formulated as (polynomially) solvable cases of the traveling salesman problem (TSP), we provide

efficient exact solutions to these scheduling problems. An in-depth analysis of the newly defined cases of TSP is provided in Chapter 5.

Chapter 4 investigates the cycle time minimization problem in a two-machine job shop, where each job consists of at most three operations. The problem was posed by Hall, Lee, and Posner (2002). They presented an $O(n^{13}p_{\max}^{11})$ algorithm to solve a special case. In this chapter, we reduce the problem to a two-machine reentrant flow shop problem. By extending previous results on the reentrant flow shop problem, we propose an algorithm that runs in $O(n^6p_{\max}^4)$ time and $O(n^4p_{\max}^3)$ space for some new cases. We also give an FPTAS for these cases. Furthermore, we identify several well-solvable special cases and present a fast 4/3-approximation algorithm for the general problem.

Chapter 5 presents collaborative work with Vladimir G. Deĭneko. In this chapter, we study special cases of the traveling salesman problem on permuted Monge matrices, which arose from robotic-cell scheduling problems. By using the theory of subtour patching, we reduce the problems to finding a minimum-b-weight spanning tree in the patching graph. In general, this problem is $\mathcal{NP}$-hard. We show, however, that newly defined special properties of the distance matrix allow us to find in polynomial time a minimum-b-weight spanning tree, and thus an optimal tour, for these new classes.

Finally, Chapter 6 draws out the overall implications of the research and poses some questions for future research.

Chapter 2
Scheduling Reentrant Robotic Cells

2.1 Introduction

Cellular manufacturing is widely used in modern production systems. The idea behind cellular manufacturing is to group similar parts together to be produced in a specialized and integrated manufacturing cell. Typically, the manufacturing cell consists of a small number of versatile machines that can perform a variety of tasks. To realize full automation and increase efficiency, these manufacturing cells are usually served by a single material handling device such as a robot, an Automated Guided Vehicle (AGV), a crane, or a hoist, which is used to load, unload, and move parts between machines. When the device is a robot, these automated systems are often called *robotic cells*. For a robotic cell, there are a number of issues to be considered, e.g., cell design, robot movement, part processing sequence, to name just a few. In this chapter, we focus on the scheduling issues. Observing that most frequently there is only one robot (material handling device) in the normal setting, the robot is often the bottleneck of systems. Therefore, unlike traditional scheduling models in which we are looking for the processing sequence of parts, *both* the robot activities and the part processing sequence should be considered in robotic cells.

Research on scheduling and sequencing in robotic cells has received much attention in the literature in recent years. The research falls into two streams. One stream is devoted to traditional production with the makespan objective. Kise, Shioyama, and Ibaraki (1991) study a two-machine no-buffer manufacturing cell served by a single AGV, which is equivalent to a two-machine robotic cell. An $O(n^2 \log n)$ algorithm is given to find the part processing sequence to minimize the makespan under a given AGV move cycle for n parts. When the manufacturing cell has unlimited buffer, Kise (1991) shows that the part sequencing problem is $\mathcal{NP}$-hard. Hurink and Knust (2001) strengthen this complexity result by showing that the problem is strongly $\mathcal{NP}$-hard. They also give complexity results for other special cases of two-, three- and m-machine robotic cells with unlimited buffer. Levner, Kogan, and Maimon (1995) study a two-machine robotic cell with part-dependent transportation and setup effects. They derive an $O(n^3)$ algorithm to find the part processing sequence

minimizing the completion time of n parts under a given robot move cycle.

Another stream of research focuses on cyclic production with the cycle time objective. In a two-machine robotic cell, Sethi et al. (1992) show that there exist two potentially optimal robot move cycles that produce one unit. When the parts in an MPS are identical, they provide sufficient conditions under which each of the two cycles is optimal over all possible cycles. When the parts in an MPS are different, they show that the part sequencing problem for a given 1-unit robot move cycle can be solved in polynomial time. Hall, Kamoun, and Sriskandarajah (1997) provide an $O(n^4)$ algorithm that simultaneously optimizes the robot move cycle and part processing sequence. Aneja and Kamoun (1999) improve this and present an $O(n \log n)$ algorithm. In a three-machine robotic cell, Sethi et al. (1992) identify six potentially optimal robot move cycles that produce one unit. When the parts in an MPS are identical, they construct a decision tree for determining the best cycle among these 1-unit cycles. When the parts in an MPS are different, Hall, Kamoun, and Sriskandarajah (1997, 1998) show that, four out of the six cycles lead to efficiently solvable part sequencing problems. For the other two cycles, the part sequencing problem is shown to be strongly $\mathcal{NP}$-hard. They also prove that the general part sequencing problem not restricted to any robot move cycle is intractable as well. In an m-machine robotic cell, Sethi et al. (1992) establish that there are exactly $m!$ potentially optimal robot move cycles that produce one unit. When the parts in an MPS are identical, Crama and van de Klundert (1997) present an $O(m^3)$ dynamic programming algorithm to minimize the cycle time of an MPS over all 1-unit cycles. Given a 1-unit robot move cycle with processing time windows, Levner and Kats (1998) show that the minimum cycle time problem can be formulated as a parametric critical path problem, and thus the minimum cycle time can be found in $O(m^3)$ time. Ioachim, Sanlaville, and Lefebvre (2001) propose an $O(qm^3)$ algorithm to compute the cycle time for a given robot move cycle that produces q units. They also develop a dynamic programming algorithm to identify the optimal q-unit robot move cycle. Moreover, they show that the results can be extended to the case of different parts in an MPS with a given part processing sequence. When the parts in an MPS are different, Sriskandarajah, Hall, and Kamoun (1998) prove that the part sequencing problems associated with $2m-2$ of the $m!$ 1-unit cycles are polynomially solvable, while the part sequencing problems associated with the remaining cycles are strongly $\mathcal{NP}$-hard. For a general survey on cyclic scheduling in robotic cells, we refer the reader to Crama et al. (2000) and van de Klundert (1996).

In the above robotic-cell scheduling problems, parts are processed on machines as in a flow shop. In this chapter, we consider a more general type of robotic cell, in which parts are required to enter a certain machine or a set of machines more than once before being completed. This type of processing is called *reentrant processing*, and is common in semiconductor manufacturing and flexible machining systems, for example, the assembly of printed circuit boards (Noble 1989) and wafer fabrication

(Graves et al. 1983; Elliott 1989). Since Graves et al. (1983) introduced reentrant processing in a flow shop, many researchers have studied this type of manufacturing system. Lev and Adiri (1984) deal with a *V-shop* in which parts visit machines following the sequence $M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_{m-1} \rightarrow M_m \rightarrow M_{m-1} \rightarrow \cdots \rightarrow M_2 \rightarrow M_1$. Wang, Sethi, and van de Velde (1997) examine a *chain-reentrant shop* in which each part has the processing route $M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_m \rightarrow M_1$. Middendorf and Timkovsky (2002) call this a *loop-reentrant shop*. Kubiak, Lou, and Wang (1996) study a *hub-reentrant shop* where parts follow the route $M_1 \rightarrow M_2 \rightarrow M_1 \rightarrow M_3 \rightarrow \cdots \rightarrow M_1 \rightarrow M_m \rightarrow M_1$, i.e., parts enter the hub machine, M_1, and other machines in alternating fashion. In general, there are numerous types of reentrant processing possible depending on the processing route, see Middendorf and Timkovsky (2002) for an extensive survey.

The remainder of the chapter is organized as follows. Section 2.2 contains definitions and notation. In Section 2.3, we present polynomial-time solutions for the cycle time and the makespan minimization problems in a two-machine reentrant robotic cell. In Section 2.4, we explore the potentially optimal robot move cycles in a three-machine loop-reentrant robotic cell. We study the cycle time minimization problem under single cycles and mixed cycles. We characterize the easily solvable and computationally hard cases. Section 2.5 contains our concluding remarks.

2.2 Definitions and Notation

There are m machines $M_1, \ldots, M_m$, $m \geq 2$, in an automated manufacturing cell served by a robot. Due to space limitations and the compact design of the cell, the machines have no input or output buffer. All parts are available at an input station (In) at time zero. The robot picks up each part at In, moves it to the first machine and loads it on that machine for processing; after the processing is completed, the robot unloads the part and moves it to the next machine for processing, and so on. After the part completes all of its processing by machines, the robot moves the finished part to an output station (Out) and drops it there. The input and output stations have unlimited storage capacity. In, $M_1, \ldots, M_m$, and Out are located on the arc of a circle or on a straight line with the robot at the center. Figure 2.1 illustrates a circle layout of such robot-centered cell with m machines.

In classical robotic cells, each part being processed passes through the machines over the same route $M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_m$. An extension of this is to allow some of the machines to be visited more than once while all parts still follow the same processing route. When a part returns to a machine for processing we say that it requires *reentrant processing*. Consequently, a robotic cell with reentrant processing is called a *reentrant robotic cell*. Without loss of generality, we assume that the consecutive operations of a part have to be performed by different machines. Both the operations of the robot and the processing of parts on machines are nonpreemptive.

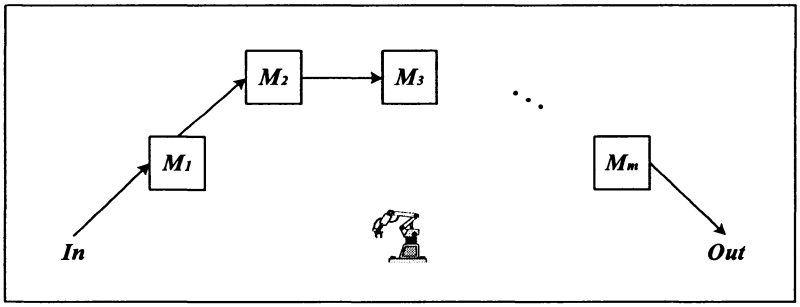

Figure 2.1: An m-machine robotic cell

Since there is no buffer space on machines, any part being produced must be either on one of the machines or on the robot. Neither a machine nor the robot can handle more than one part at a time.

A robot move cycle is a sequence of robot moves that returns to the initial state. A 1-unit cycle returns to the same state after the production of a single unit. Recognizing that a robotic cell is a discrete dynamic system, Sethi et al. (1992) define the robot move cycle as a sequence of discrete state transitions. We extend their representation to accommodate reentrant processing. More specifically, a state of the system is defined by the $(m+1)$-tuple $\mathcal{G} = (g_1, g_2, \ldots, g_m, g_{m+1})$, where $g_i \in \{\varnothing, \Omega\}$, $i = 1, 2, \ldots, m$, and $g_{m+1} \in \{I, O, M_1^{(h)-}, \ldots, M_m^{(h)-}, M_1^{(h)+}, \ldots, M_m^{(h)+}\}$. Here $g_i = \varnothing$ or Ω means that M_i is not occupied or is occupied by a part, respectively; g_{m+1} refers to the robot position with I denoting the robot at In just as it arrives there and begins to pick up a part, O denoting the robot just as it completes the dropping of a part at Out and begins to leave there, $M_i^{(h)-}$ denoting the robot at M_i just after it has finished loading a part on the machine for the hth time, $M_i^{(h)+}$ denoting the robot at M_i just before it begins unloading a part from the machine the hth time. The h superscript is omitted if the part is processed on a machine just once.

We use the following notation to describe a reentrant robotic cell.

n	=	the total number of parts to be produced or the number of parts in an MPS.
σ	=	the processing sequence of parts, $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$.
$P_{\sigma(k)}$	=	the kth part in the sequence σ to be produced, $k = 1, 2, \ldots, n$.
a_{ih}, b_{ih}, c_{ih}	=	the processing times of part i on machines M_1, M_2 and M_3 for the hth time, respectively. (For a_{ih}, b_{ih} and c_{ih}, the subscript h is omitted if the part is processed just once.)
δ	=	the time taken by the robot to move between adjacent location pairs $(In, M_1), (M_i, M_{i+1})$, $i = 1, \ldots, m-1$, and (M_m, Out). Travel between any two locations is via intermediate locations. Thus, e.g., the movement from In to Out takes $(m+1)\delta$ time.
ε	=	the time needed to pick up, (un)load or drop a part by the robot.

Throughout this chapter, all numbers are assumed to be non-negative integers, and we restrict our analysis to 1-unit robot move cycles. For ease of exposition, we assume that the travel, pick up, load, unload, and drop times are independent of the machines, but all results in this chapter are also applicable to machine-dependent travel times and (un)loading times.

Following the three-field notation $\alpha|\beta|\gamma$ for machine scheduling problems, besides Fm for flow shops, we use RCm and $RRCm$ under α to indicate a regular m-machine robotic cell and an m-machine reentrant robotic cell, respectively. Under β, we may have S_i to express that robot move cycle S_i alone is used. When S_i is omitted, any cycle may be used. Also, we use $\ell = 2K$ or $\ell = 2K+1$ under β to describe the even or odd number of operations for a part, respectively. Under γ, we may have $C_{\max}$ and C_t to denote the makespan and cycle time objectives, respectively. For example, $RRC2|S_2|C_t$ denotes the problem of minimizing the cycle time for producing MPSs in a two-machine reentrant robotic cell, restricted to robot move cycle S_2.

2.3 Two-machine Reentrant Cells

In any two-machine reentrant robotic cell, with the appropriate numbering of the machines, the part processing route is always $M_1 \rightarrow M_2 \rightarrow M_1 \rightarrow M_2 \rightarrow \cdots$. Let ℓ be the number of operations for a part. Since ℓ may be even or odd, we distinguish two cases for the $RRC2||C_t$ and $RRC2||C_{\max}$ problems: $\ell = 2K$ and $\ell = 2K+1$, where $K \geq 1$. We start with the $\ell = 2K+1$ case.

2.3.1 Odd number of operations

When $\ell = 2K+1$, each part is processed $K+1$ times on machine M_1 and K times on machine M_2. In this case, only one robot move cycle is feasible. Let us consider the system in the initial state $\mathcal{G}$ when the robot just arrives at In and begins to pick up a part, i.e., $\mathcal{G} = (\varnothing, \varnothing, I)$. The robot move cycle can be described by the following sequence of states:

$$(\varnothing,\varnothing,I),\underbrace{(\Omega,\varnothing,M_1^{(h)-}),(\Omega,\varnothing,M_1^{(h)+}),(\varnothing,\Omega,M_2^{(h)-}),(\varnothing,\Omega,M_2^{(h)+})}_{\text{repeat } K \text{ times for } h=1,2,\ldots,K},$$
$$(\Omega,\varnothing,M_1^{(K+1)-}),(\Omega,\varnothing,M_1^{(K+1)+}),(\varnothing,\varnothing,O),(\varnothing,\varnothing,I).$$

More precisely,

the activities and their times in this cycle are: Pick up part $P_{\sigma(k)}$:(ε), move it to M_1:(δ), repeat K times the activities in the following braces {load the part on M_1:(ε), wait for its processing: ($a_{\sigma(k)h}$), unload the part:(ε), move it to M_2:(δ) and load it on M_2:(ε), wait for its processing:($b_{\sigma(k)h}$), unload the part:(ε), then move it

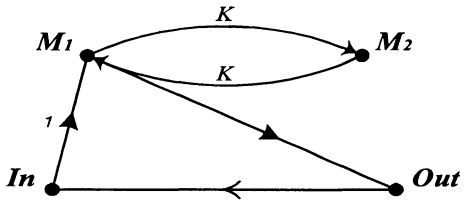

Figure 2.2: Two-machine cell with $2K+1$ operations

back to M_1:(δ), where $h = 1, 2, \ldots, K\}$, after this, load part $P_{\sigma(k)}$ on M_1 again:(ε), wait for its processing:($a_{\sigma(k)(K+1)}$), unload the part:(ε), move it from M_1 to *Out*:(2δ) and drop it there:(ε), then move to *In*:(3δ). See Figure 2.2 for a diagram. (In the figure the arc next to "1" symbolizes the initial robot move; the K stands for the number of robot moves between machines.)

The time required by this cycle can be calculated as

$$\begin{aligned} T_{\sigma(k)} &= \varepsilon + \delta + \sum_{h=1}^{K} (\varepsilon + a_{\sigma(k)h} + \varepsilon + \delta + \varepsilon + b_{\sigma(k)h} + \varepsilon + \delta) + (\varepsilon + a_{\sigma(k)(K+1)} + \varepsilon) \\ &\quad + 2\delta + \varepsilon + 3\delta \\ &= (4 + 4K)\varepsilon + (6 + 2K)\delta + \sum_{h=1}^{K+1} a_{\sigma(k)h} + \sum_{h=1}^{K} b_{\sigma(k)h}. \end{aligned}$$

It is easy to see that the (cycle) time for producing an MPS in *any* sequence is given by $C_t = \sum_{i=1}^{n}((4+4K)\varepsilon + (6+2K)\delta + \sum_{h=1}^{K+1} a_{ih} + \sum_{h=1}^{K} b_{ih})$, which is independent of the part processing sequence. The makespan for producing the n parts in the MPS can be derived from C_t by removing the time needed to re-position the robot from *Out* to *In* at the end, i.e., $C_{\max} = C_t - 3\delta$, which is independent of the part processing sequence as well.

2.3.2 Even number of operations

When $\ell = 2K$, each part has to be processed K times on both machines M_1 and M_2. This kind of repeated processing frequently occurs in paint shops, where a part requires a repetitive painting and baking of several coats of paint before it is finished. Note that when $K = 1$, the problem reduces to a regular (no reentry) two-machine bufferless robotic cell, which was studied in Kise, Shioyama, and Ibaraki (1991), Sethi et al. (1992), Hall, Kamoun, and Sriskandarajah (1997), as well as Aneja and Kamoun (1999).

Let us consider the system in state $\mathcal{G}$ when the robot has just loaded a part on machine M_2 the Kth time, i.e., $\mathcal{G} = (\varnothing, \Omega, M_2^{(K)-})$. We show that, similarly to the regular robotic cell with no reentry, there are two feasible robot move cycles to

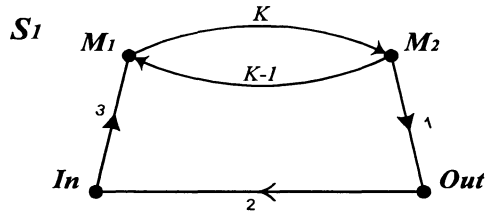

Figure 2.3: The robot move cycle S_1

continue from state $\mathcal{G}$, S_1 and S_2—shown in Figures 2.3 and 2.4. (In these figures the numbers $1, 2, \ldots$ show the sequence of the initial steps, whereas K and $K-1$ represent the number of a part's moves between machines.) Sethi et al. (1992) have analyzed the cycle times for the two feasible cycles in the regular robotic cell. In the following, we show how this can be extended to the $K > 1$ case.

Lemma 2.1 *There are only two robot move cycles (S_1 and S_2) which could minimize the cycle time for producing MPSs.*

Proof. The robot cannot move to In to pick up a new part (and then load it on M_1) before the current part is loaded on machine M_2 the Kth time (i.e., state $\mathcal{G}$). For otherwise, it would lead to a deadlock, where both machines are loaded with a part and both parts are waiting for the robot to move them to the next machine. Thus $\mathcal{G}$ is the only state in which we have a choice for the next robot move. At this state, the robot can either move to In to pick up a new part (cycle S_2) or wait for the processing of the current part to be finished (cycle S_1). □

Cycles S_1 and S_2 share the common state $(\varnothing, \Omega, M_2^{(K)-})$. It is clear that switching from S_1 to S_2 or vice versa without wasteful robot moves is possible only through this state. We will assume without loss of generality that both cycles start at this state. Then the cycles S_1 and S_2 can be described as

$$S_1\colon (\varnothing, \Omega, M_2^{(K)-}), (\varnothing, \Omega, M_2^{(K)+}), (\varnothing, \varnothing, O), (\varnothing, \varnothing, I), (\Omega, \varnothing, M_1^{(1)-}), (\Omega, \varnothing, M_1^{(1)+}),$$
$$\underbrace{(\varnothing, \Omega, M_2^{(h)-}), (\varnothing, \Omega, M_2^{(h)+}), (\Omega, \varnothing, M_1^{(h+1)-}), (\Omega, \varnothing, M_1^{(h+1)+})}_{\text{repeat } K-1 \text{ times for } h=1,2,\ldots,K-1}, (\varnothing, \Omega, M_2^{(K)-})$$

and

$$S_2\colon (\varnothing, \Omega, M_2^{(K)-}), (\varnothing, \Omega, I), (\Omega, \Omega, M_1^{(1)-}), (\Omega, \Omega, M_2^{(K)+}), (\Omega, \varnothing, O), (\Omega, \varnothing, M_1^{(1)+}),$$
$$\underbrace{(\varnothing, \Omega, M_2^{(h)-}), (\varnothing, \Omega, M_2^{(h)+}), (\Omega, \varnothing, M_1^{(h+1)-}), (\Omega, \varnothing, M_1^{(h+1)+})}_{\text{repeat } K-1 \text{ times for } h=1,2,\ldots,K-1}, (\varnothing, \Omega, M_2^{(K)-}).$$

Let $\mu = 4\varepsilon + 6\delta$, $B_{\sigma(k)} = b_{\sigma(k)K} + 2\varepsilon + 2\delta$, and $A_{\sigma(k+1)} = a_{\sigma(k+1)1} + 2\varepsilon + 2\delta$. To determine the time required by each cycle, let us examine now their activities. Without loss of generality, we assume that the current part is $P_{\sigma(k)}$.

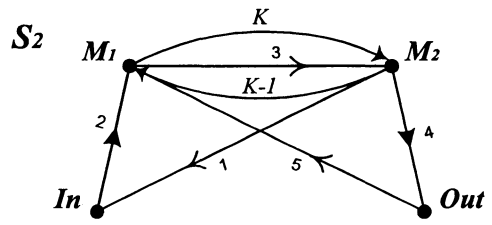

Figure 2.4: The robot move cycle S_2

The activities and their times in robot move cycle S_1 ***are:*** Wait for part $P_{\sigma(k)}$ to finish its processing on M_2:$(b_{\sigma(k)K})$, when it is finished, unload the part:(ε), move it to *Out*:(δ) and drop it there:(ε), move to *In*:(3δ), pick up the next part, $P_{\sigma(k+1)}$:(ε), move it to M_1:(δ) and load it on M_1:(ε), wait for its processing to be finished:$(a_{\sigma(k+1)1})$, unload the part:(ε), move it to M_2:(δ), after this, repeat $K-1$ times the following subcycle of activities in braces {load part $P_{\sigma(k+1)}$ on M_2:(ε), wait for its processing to be finished:$(b_{\sigma(k+1)h})$, unload it:(ε), move it back to M_1:(δ), load it again on M_1:(ε), wait for its processing:$(a_{\sigma(k+1)(h+1)})$, unload the part:(ε), move it to M_2:(δ), where $h = 1, 2, \ldots, K-1$}, finally, load part $P_{\sigma(k+1)}$ on M_2:(ε).

Then the time required by cycle S_1 can be derived as

$$\begin{aligned}
T^1_{\sigma(k)\sigma(k+1)} &= b_{\sigma(k)K} + \varepsilon + \delta + \varepsilon + 3\delta + \varepsilon + \delta + \varepsilon + a_{\sigma(k+1)1} + \varepsilon + \delta \\
&\quad + \sum_{h=1}^{K-1} (\varepsilon + b_{\sigma(k+1)h} + \varepsilon + \delta + \varepsilon + a_{\sigma(k+1)(h+1)} + \varepsilon + \delta) + \varepsilon \\
&= 2\varepsilon + 2\delta + (b_{\sigma(k)K} + 2\varepsilon + 2\delta) + (a_{\sigma(k+1)1} + 2\varepsilon + 2\delta) \\
&\quad + \sum_{h=1}^{K-1} (4\varepsilon + 2\delta + a_{\sigma(k+1)(h+1)} + b_{\sigma(k+1)h}) \\
&= (4K-2)\varepsilon + 2K\delta + \sum_{h=1}^{K-1} (a_{\sigma(k+1)(h+1)} + b_{\sigma(k+1)h}) \\
&\quad + B_{\sigma(k)} + A_{\sigma(k+1)}. \qquad (2.1)
\end{aligned}$$

The activities and their times in robot move cycle S_2 ***are:*** Move to *In*:(2δ), pick up the next part, $P_{\sigma(k+1)}$:(ε), move it to M_1:(δ) and load it on M_1:(ε), move to M_2:(δ), if needed, wait for part $P_{\sigma(k)}$ on M_2:$(w_2(k))$, unload the part:(ε), move it to *Out*:(δ) and drop it there:(ε), move to M_1:(2δ), if needed, wait for part $P_{\sigma(k+1)}$ to be finished on M_1:$(w_1(k+1))$, unload the part:(ε), move it to M_2:(δ), after this, repeat $K-1$ times the following subcycle of activities in braces {load part $P_{\sigma(k+1)}$ on M_2:(ε), wait for its processing:$(b_{\sigma(k+1)h})$, unload it:(ε), move it back to M_1:(δ), load it again on M_1:(ε), wait for its processing:$(a_{\sigma(k+1)(h+1)})$, unload the part:(ε), move it to M_2:(δ), where $h = 1, 2, \ldots, K-1$}, finally, load part $P_{\sigma(k+1)}$ on M_2:(ε).

Then the time required by cycle S_2 can be derived as

$$\begin{aligned} T^2_{\sigma(k)\sigma(k+1)} &= 2\delta+\varepsilon+\delta+\varepsilon+\delta+w_2(k)+\varepsilon+\delta+\varepsilon+2\delta+w_1(k+1)+\varepsilon+\delta \\ &\quad+\sum_{h=1}^{K-1}(\varepsilon+b_{\sigma(k+1)h}+\varepsilon+\delta+\varepsilon+a_{\sigma(k+1)(h+1)}+\varepsilon+\delta)+\varepsilon \\ &= 6\varepsilon+8\delta+w_1(k+1)+w_2(k)+\sum_{h=1}^{K-1}(4\varepsilon+2\delta+a_{\sigma(k+1)(h+1)}+b_{\sigma(k+1)h}), \end{aligned}$$

where $w_1(k+1)=\max\{0,\ a_{\sigma(k+1)1}-(\delta+w_2(k)+\varepsilon+\delta+\varepsilon+2\delta)\}=\max\{0,\ a_{\sigma(k+1)1}-(2\varepsilon+4\delta+w_2(k))\}$ is the robot waiting time at M_1 for the 1st processing of part $P_{\sigma(k+1)}$, and $w_2(k)=\max\{0,\ b_{\sigma(k)K}-(2\varepsilon+4\delta)\}$ is the robot waiting time at M_2 for the Kth processing of part $P_{\sigma(k)}$.

Adding $w_1(k+1)$ and $w_2(k)$ and substituting for $w_2(k)$, $T^2_{\sigma(k)\sigma(k+1)}$ may be simplified as

$$\begin{aligned} T^2_{\sigma(k)\sigma(k+1)} &= 6\varepsilon+8\delta+\max\{w_2(k),\ a_{\sigma(k+1)1}-(2\varepsilon+4\delta)\} \\ &\quad+\sum_{h=1}^{K-1}(4\varepsilon+2\delta+a_{\sigma(k+1)(h+1)}+b_{\sigma(k+1)h}) \\ &= 6\varepsilon+8\delta+\max\{\max\{0,\ b_{\sigma(k)K}-(2\varepsilon+4\delta)\},\ a_{\sigma(k+1)1}-(2\varepsilon+4\delta)\} \\ &\quad+\sum_{h=1}^{K-1}(4\varepsilon+2\delta+a_{\sigma(k+1)(h+1)}+b_{\sigma(k+1)h}) \\ &= 2\varepsilon+2\delta+\max\{4\varepsilon+6\delta,\ b_{\sigma(k)K}+(2\varepsilon+2\delta),\ a_{\sigma(k+1)1}+(2\varepsilon+2\delta)\} \\ &\quad+\sum_{h=1}^{K-1}(4\varepsilon+2\delta+a_{\sigma(k+1)(h+1)}+b_{\sigma(k+1)h}) \\ &= (4K-2)\varepsilon+2K\delta+\sum_{h=1}^{K-1}(a_{\sigma(k+1)(h+1)}+b_{\sigma(k+1)h}) \\ &\quad+\max\{\mu, B_{\sigma(k)}, A_{\sigma(k+1)}\}. \end{aligned} \tag{2.2}$$

Observe that both $T^1_{\sigma(k)\sigma(k+1)}$ and $T^2_{\sigma(k)\sigma(k+1)}$ have the term $\sum_{h=1}^{K-1}(a_{\sigma(k+1)(h+1)}+b_{\sigma(k+1)h})$ and the constant term $(4K-2)\varepsilon+2K\delta$. The difference between them is that $T^1_{\sigma(k)\sigma(k+1)}$ has the term $B_{\sigma(k)}+A_{\sigma(k+1)}$, while $T^2_{\sigma(k)\sigma(k+1)}$ has the term $\max\{\mu, B_{\sigma(k)}, A_{\sigma(k+1)}\}$.

Cycle time minimization

Hall, Kamoun, and Sriskandarajah (1997) have proved that a single robot move cycle is not optimal, in general, for the cycle time minimization problem in the regular

robotic cell. They have given an example to show that the cycle time is minimized by a mix of robot move cycles S_1 and S_2. This observation clearly remains applicable for our problem, $RRC2|\ell = 2K|C_t$, too. We can give sufficient conditions, however, which guarantee that a single robot move cycle is optimal for our problem. Let us assume in the remainder of the chapter that the b_{iK}'s are in non-decreasing order, i.e., $b_{iK} \leq b_{(i+1)K}$, and let θ be an assignment (ordering) for which $a_{\theta(i)1} \leq a_{\theta(i+1)1}$. Thus $b_{iK} + a_{\theta(i)1} \leq b_{(i+1)K} + a_{\theta(i+1)1}$ for $i = 1, 2, \ldots, n-1$.

Theorem 2.2 *The cycle time for producing MPSs is minimized under cycle S_1 if $b_{nK} + a_{\theta(n)1} \leq 2\delta$, and under cycle S_2 if $b_{1K} + a_{\theta(1)1} \geq 2\delta$.*

Proof. Since the difference between $T^1_{\sigma(k)\sigma(k+1)}$ and $T^2_{\sigma(k)\sigma(k+1)}$ is that $T^1_{\sigma(k)\sigma(k+1)}$ has the term $B_{\sigma(k)} + A_{\sigma(k+1)}$ while $T^2_{\sigma(k)\sigma(k+1)}$ has the term $\max\{\mu, B_{\sigma(k)}, A_{\sigma(k+1)}\}$, it suffices to compare these two terms.

- If $b_{nK}+a_{\theta(n)1} \leq 2\delta \Rightarrow B_{\sigma(k)}+A_{\sigma(k+1)} \leq B_n+A_{\theta(n)} \leq 4\varepsilon+6\delta = \mu \Rightarrow T^1_{\sigma(k)\sigma(k+1)} \leq T^2_{\sigma(k)\sigma(k+1)} \Rightarrow$ cycle S_1 is optimal;
- If $b_{1K}+a_{\theta(1)1} \geq 2\delta \Rightarrow B_{\sigma(k)}+A_{\sigma(k+1)} \geq B_1+A_{\theta(1)} \geq 4\varepsilon+6\delta = \mu \Rightarrow T^1_{\sigma(k)\sigma(k+1)} \geq T^2_{\sigma(k)\sigma(k+1)} \Rightarrow$ cycle S_2 is optimal. □

Remark 1 Note that the sufficient conditions in Theorem 2.2 are not dependent on ε. Intuitively, the theorem states that if the processing times are all small compared to δ, then it is always better to wait for the current part to finish its processing (cycle S_1). On the other hand, if the processing times are all large in comparison to δ, then cycle S_2 is better, i.e., the robot should load the next part on M_1 before the current part is finished on M_2.

Remark 2 Solving the $RRC2|\ell = 2K|C_t$ problem in general requires determining the optimal sequence of the S_1 and S_2 cycles to be applied to the n parts in an MPS. If this sequence consists of repeating a subsequence of q cycles n/q times (q is a divisor of n by assumption), then we say that the policy of repeating a q-unit cycle is optimal. Sethi et al. (1992) have shown that the repetition of the best 1-unit cycle dominates all other policies for an MPS with *identical* parts in the regular two-machine robotic cell, i.e., it will yield the maximum throughput rate. As $a_{11} = a_{21} = \cdots = a_{n1} = a$ and $b_{1K} = b_{2K} = \cdots = b_{nK} = b$ for identical parts, we can see from the above theorem that using the best 1-unit cycle is also optimal for our problem in the case of an MPS with identical parts.

For the remainder of this section, we consider the general case of our problem when the sufficient conditions of Theorem 2.2 are not satisfied, i.e., both cycles S_1 and S_2 may have to be used to find the minimum cycle time.

Theorem 2.3 *The problem of finding the optimal robot move cycle and part processing sequence to minimize the cycle time in the $RRC2|\ell = 2K|C_t$ problem can be formulated as a traveling salesman problem (TSP).*

Proof. As stated in Lemma 2.1, at the state $\mathcal{G} = (\varnothing, \Omega, M_2^{(K)-})$, we either wait for the processing of the current part (cycle S_1) or move to In to pick up a new part (cycle S_2). The elapsed time between two consecutive occurrences of state $\mathcal{G}$ involving the processing of parts $P_{i=\sigma(k)}$ and $P_{j=\sigma(k+1)}$ will be

$$\begin{aligned} T_{\sigma(k)\sigma(k+1)} &= \min\{T^1_{\sigma(k)\sigma(k+1)}, T^2_{\sigma(k)\sigma(k+1)}\} \\ &= \Delta + \Gamma(j) + \min\{B_i + A_j, \max\{\mu, B_i, A_j\}\}, \end{aligned} \tag{2.3}$$

where $\Delta = (4K-2)\varepsilon + 2K\delta$, $\Gamma(j) = \sum_{h=1}^{K-1}(a_{j(h+1)} + b_{jh})$, $\mu = 4\varepsilon + 6\delta$, $B_i = b_{iK} + 2\varepsilon + 2\delta$, and $A_j = a_{j1} + 2\varepsilon + 2\delta$.

However, since Δ is constant and $\Gamma(j)$ is independent of i, the elapsed time between the two consecutive states $\mathcal{G}$ can be viewed as the sum of terms independent of i plus the distance between city i and j in a TSP with distance matrix $C = (c_{ij})$, where $c_{ij} = \min\{B_i + A_j, \max\{\mu, B_i, A_j\}\}$. As a result, the scheduling problem is equivalent to the problem of finding an optimal tour for this n-city TSP. Suppose that the optimal tour (part processing sequence) is $\sigma^*(1), \ldots, \sigma^*(n), \sigma^*(n+1) = \sigma^*(1)$, the minimum (cycle) time for producing an MPS is then equal to $C_t(\sigma^*) = \sum_{k=1}^{n} T_{\sigma^*(k)\sigma^*(k+1)} = n\Delta + \sum_{j=1}^{n}\Gamma(j) + \sum_{k=1}^{n} c_{\sigma^*(k)\sigma^*(k+1)}$. □

Recall that, based on the well-known Gilmore–Gomory algorithm (Gilmore and Gomory 1964) for a special case of the TSP, Aneja and Kamoun (1999) have described an $O(n \log n)$ algorithm to solve the special case of the TSP with distance matrix $C = (c_{ij}) = \min\{B_i + A_j, \max\{\mu, B_i, A_j\}\}$. Therefore, we have the following corollary.

Corollary 2.4 *The $RRC2|\ell = 2K|C_t$ problem can be solved optimally in $O(n \log n)$ time.*

In Chapter 5, we will generalize and substantially extend the aforementioned class of distance matrices, and provide a unified polynomial solution for the TSP on the extended class.

Makespan minimization

We consider now the makespan version of the reentrant robotic-cell problem, which is denoted by $RRC2|\ell = 2K|C_{\max}$. Beginning from time zero, the makespan of a schedule is the time at which the last finished part is dropped at the output station. Here it is assumed that the robot is available at the input station at time

Table 2.1: An Example in Lemma 2.5

i	1	2	3	4	5
a_i	5	860	5	860	5
b_i	860	60	860	5	5
A_i	45	900	45	900	45
B_i	900	100	900	45	45
A'_i	100	900	100	900	100

zero. Given the similarity between the cycle time and the makespan objectives, one natural question arises: does the part processing sequence that minimizes the cycle time of an MPS also minimize the makespan? We will show that the answer is negative, which makes it necessary to find a different solution for the makespan minimization problem.

Recall that Kise, Shioyama, and Ibaraki (1991) have studied a two-machine manufacturing cell served by an Automated Guided Vehicle (AGV), which is equivalent to a regular two-machine robotic cell. They have presented an algorithm to find the part processing sequence to minimize the makespan if the AGV (robot) move cycle is fixed at S_2, i.e., the $RC2|S_2|C_{\max}$ problem. Here we show first that this policy is suboptimal, i.e., the makespan can be improved by a *mix* of cycles S_1 and S_2. Moreover, instead of developing the recursive relations used in Kise, Shioyama, and Ibaraki (1991) to calculate the makespan for a given part processing sequence, we will show that the makespan can be derived directly from a solvable case of the TSP problem.

Lemma 2.5 *Following only a single robot move cycle (S_1 or S_2) is not necessarily optimal for the $RRC2|\ell = 2K|C_{\max}$ problem even with $K = 1$.*

Proof. We prove it by an example. Consider the following instance of the $RRC2|\ell = 2K|C_{\max}$ problem: $K = 1$, $n = 5$, $\varepsilon = \delta = 10$; the a_i-s and b_i-s are shown in Table 2.1. Thus $\mu = 4\varepsilon + 6\delta = 100$ and $\Delta = 2\varepsilon + 2\delta = 40$. The $A_i = a_i + 2\varepsilon + 2\delta$ and $B_i = b_i + 2\varepsilon + 2\delta$ values are also shown in Table 2.1.

As mentioned earlier, the minimum makespan under robot move cycle S_1 is independent of the part processing sequence and can be calculated as the minimum cycle time under S_1 minus the robot move time from *Out* to *In*, i.e., $C^1_{\max} = n\Delta + \sum_{i=1}^{n}(A_i+B_i)-3\delta = 200+(45+900+45+900+45)+(900+100+900+45+45)-30 = 4095$.

For a part processing sequence $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$, the makespan under robot move cycle S_2, $C^2_{\max}(\sigma)$, is equal to the sum of the initial time needed to get part $P_{\sigma(1)}$ into the first occurrence of state $\mathcal{G} = (\varnothing, \Omega, M_2^-)$, plus the sum of $T^2_{\sigma(k)\sigma(k+1)}$ for $k = 1, \ldots, n-1$, plus the time needed after the last occurrence of state $\mathcal{G}$ to get part $P_{\sigma(n)}$ finished and delivered to *Out*. Let $A'_i = \max\{\mu, A_i\}$. (The A'_i values for the

example are shown in the last row of Table 2.1.) $C^2_{\max}(\sigma)$ may be derived as follows:

$$\begin{aligned} C^2_{\max}(\sigma) &= \varepsilon + \delta + \varepsilon + a_{\sigma(1)} + \varepsilon + \delta + \varepsilon + \sum_{k=1}^{n-1} \left(2\varepsilon + 2\delta + \max\{\mu, B_{\sigma(k)}, A_{\sigma(k+1)}\}\right) \\ &\quad + b_{\sigma(n)} + \varepsilon + \delta + \varepsilon \\ &= A_{\sigma(1)} + \sum_{k=1}^{n-1} \left(\Delta + \max\{\mu, B_{\sigma(k)}, A_{\sigma(k+1)}\}\right) + B_{\sigma(n)} + 2\varepsilon - \delta \\ &= A_{\sigma(1)} + \sum_{k=1}^{n-1} \max\{B_{\sigma(k)}, A'_{\sigma(k+1)}\} + B_{\sigma(n)} + n\Delta - 3\delta. \end{aligned} \tag{2.4}$$

To find the best part processing sequence $\widehat{\sigma}$ that minimizes $C^2_{\max}(\sigma)$, Kise, Shioyama, and Ibaraki (1991) presented the optimal algorithm Makespan-S_2, which fixes the first part in every possible way and solves an n-city TSP repeatedly.

Algorithm Makespan-S_2
Input: $A'_i = \max\{\mu, A_i\}$ and B_i, $i = 1, 2, \ldots, n$.
1. Let L be a large positive number.
2. For $r = 1, 2, \ldots, n$ do
   2a. Find the best n-city TSP tour σ' with distance matrix $C = (c_{ij}) = \max\{B_i, A'_j\}$ and $A'_r = 0$. Let the length of σ' be L'.
   2b. If $L' + A_r < L$ then $L = L' + A_r$ and $\widehat{\sigma} = \sigma'$.

Output: The best part processing sequence $\widehat{\sigma}$ and $C^2_{\max}(\widehat{\sigma}) = L + n\Delta - 3\delta$.

Using Algorithm Makespan-S_2, we can find that the part processing sequence $\widehat{\sigma} = (1, 2, 3, 4, 5)$ will minimize $C^2_{\max}(\sigma)$ for the above instance. Calculated by (2.4), the minimum makespan under S_2 is equal to $C^2_{\max}(\widehat{\sigma}) = 45 + (900 + 100 + 900 + 100) + 45 + 200 - 30 = 2260$.

Let us consider now a mix of cycles S_1 and S_2. Starting from the first occurrence of state $\mathcal{G} = (\varnothing, \Omega, M_2^-)$, let the robot follow the sequence of cycles S_2, S_2, S_2, S_1. Recall that for consecutive parts $P_{\sigma(k)}$ and $P_{\sigma(k+1)}$, the times of cycles S_1 and S_2 are given by (2.1) and (2.2), respectively. That is, $T^1_{\sigma(k)\sigma(k+1)} = \Delta + B_{\sigma(k)} + A_{\sigma(k+1)}$ and $T^2_{\sigma(k)\sigma(k+1)} = \Delta + \max\{\mu, B_{\sigma(k)}, A_{\sigma(k+1)}\} = \Delta + \max\{B_{\sigma(k)}, A'_{\sigma(k+1)}\}$ for $K = 1$. Assume the part processing sequence is $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(5)) = (1, 2, 3, 4, 5)$. Then the time between the first and the last occurrences of state $\mathcal{G}$ can be calculated as $\sum_{k=1}^{3} T^2_{\sigma(k)\sigma(k+1)} + T^1_{\sigma(4)\sigma(5)} = (40+900)+(40+100)+(40+900)+(40+45+45) = 2150$. Clearly, it takes $4\varepsilon + 2\delta + a_{\sigma(1)} = 65$ units of time from time zero to get part $P_{\sigma(1)}$ into the first occurrence of state $\mathcal{G}$. It also needs $2\varepsilon + \delta + b_{\sigma(5)} = 35$ units of time after the last occurrence of state $\mathcal{G}$ to get part $P_{\sigma(5)}$ finished and delivered to *Out*. Therefore, the makespan for producing these five parts is equal to $65 + 2150 + 35 = 2250$, which is less than the minimum makespan using only S_1 or S_2. □

As following a single robot move cycle is not optimal generally, we consider now both cycles to find the minimum makespan for our problem. The minimum elapsed time T_{ij} between two consecutive states $\mathcal{G}$ on parts $P_{i=\sigma(k)}$ and $P_{j=\sigma(k+1)}$, $k = 1, 2, \ldots, n-1$ can be calculated by (2.3). Then the makespan under part processing sequence $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$ can be calculated as the sum of the initial time needed to get part $P_{\sigma(1)}$ into the first occurrence of state $\mathcal{G}$, plus the sum of the T_{ij}'s, plus the time needed after the last occurrence of state $\mathcal{G}$ to get part $P_{\sigma(n)}$ finished and delivered to *Out*, i.e.,

$$\begin{aligned}
C_{\max}(\sigma) &= \varepsilon + \delta + \varepsilon + a_{\sigma(1)1} + \varepsilon + \delta + \varepsilon + \sum_{h=1}^{K-1}(b_{\sigma(1)h} + 2\varepsilon + \delta + a_{\sigma(1)(h+1)} + 2\varepsilon + \delta) \\
&\quad + \sum_{k=1}^{n-1} T_{\sigma(k)\sigma(k+1)} + \left(b_{\sigma(n)K} + \varepsilon + \delta + \varepsilon\right) \\
&= 6\varepsilon + 3\delta + a_{\sigma(1)1} + b_{\sigma(n)K} + \sum_{h=1}^{K-1}(4\varepsilon + 2\delta + a_{\sigma(1)(h+1)} + b_{\sigma(1)h}) \\
&\quad + \sum_{k=1}^{n-1} T_{\sigma(k)\sigma(k+1)} \\
&= 4\varepsilon + \delta + a_{\sigma(1)1} + b_{\sigma(n)K} + \Delta + \Gamma(\sigma(1)) + \sum_{k=1}^{n-1} T_{\sigma(k)\sigma(k+1)} \\
&= A_{\sigma(1)} + \sum_{k=1}^{n-1} d_{\sigma(k)\sigma(k+1)} + B_{\sigma(n)} + \sum_{k=1}^{n} \Gamma(\sigma(k)) + n\Delta - 3\delta, \qquad (2.5)
\end{aligned}$$

where $d_{\sigma(k)\sigma(k+1)} = \min\{B_{\sigma(k)} + A_{\sigma(k+1)}, \max\{\mu, B_{\sigma(k)}, A_{\sigma(k+1)}\}\}$.

Since in (2.5), $n\Delta - 3\delta$ is constant and $\sum_{k=1}^{n} \Gamma(\sigma(k))$ is independent of the part processing sequence, it is clear that minimizing the makespan $C_{\max}(\sigma)$ is equivalent to minimizing $A_{\sigma(1)} + \sum_{k=1}^{n-1} d_{\sigma(k)\sigma(k+1)} + B_{\sigma(n)}$.

We are now ready to show that the optimal part processing sequences are not the same for the C_t and $C_{\max}$ objectives.

Lemma 2.6 *A part processing sequence that is optimal for the $RRC2|\ell = 2K|C_t$ problem is not necessarily optimal for the $RRC2|\ell = 2K|C_{\max}$ problem even with $K = 1$, and vice versa.*

Proof. By Equation (2.3), minimizing the cycle time of an MPS is equivalent to minimizing $\sum_{k=1}^{n} d_{\sigma(k)\sigma(k+1)}$. Thus it suffices to show that a part processing sequence that minimizes $\sum_{k=1}^{n} d_{\sigma(k)\sigma(k+1)}$ does not necessarily minimize $A_{\sigma(1)} + \sum_{k=1}^{n-1} d_{\sigma(k)\sigma(k+1)} + B_{\sigma(n)}$, and vice versa. Note that we assume $\sigma(n+1) = \sigma(1)$ for the cycle time minimization problem.

Table 2.2: An Example in Lemma 2.6

i	1	2	3
a_i	1	1250	1197
b_i	1249	1098	1199
A_i	801	2050	1997
B_i	2049	1898	1999
A'_i	2000	2050	2000

Consider the following instance of the $RRC2|\ell = 2K|C_t$ and $RRC2|\ell = 2K|C_{\max}$ problems: $K = 1$, $n = 3$, $\varepsilon = \delta = 200$; the a_i-s and b_i-s are shown in Table 2.2. Thus $\mu = 4\varepsilon + 6\delta = 2000$. The $A_i = a_i + 2\varepsilon + 2\delta$, $B_i = b_i + 2\varepsilon + 2\delta$, and $A'_i = \max\{\mu, A_i\}$ values are also shown in Table 2.2.

As $n = 3$, there are only two different cyclic part processing sequences: $(1, 2, 3, 1, 2, 3, \ldots)$ and $(1, 3, 2, 1, 3, 2, \ldots)$. Since $\sum_{k=1}^{n} d_{\sigma(k)\sigma(k+1)}$ is equal to $2049 + 2050 + 2000 = 6099$ if $\sigma = (1, 3, 2)$ and becomes $2050 + 2000 + 2000 = 6050$ if $\sigma = (1, 2, 3)$, the optimal part processing sequence under the cycle time objective is $(1, 2, 3)$.

However, $A_{\sigma(1)} + \sum_{k=1}^{n-1} d_{\sigma(k)\sigma(k+1)} + B_{\sigma(n)}$ is equal to $801 + 2050 + 2000 + 1999 = 6850$ if $\sigma = (1, 2, 3)$ and becomes $801 + 2049 + 2050 + 1898 = 6798$ if $\sigma = (1, 3, 2)$. Thus it is clear that $(1, 2, 3)$ is *not* the optimal part processing sequence under the makespan objective. From the six permutations of the part processing sequence, it is not difficult to verify that the optimal one is $(1, 3, 2)$. □

The consequence of the above lemma is that we cannot simply apply the solution method for a cycle time minimization problem to solve the makespan version of the problem. Nevertheless, we have shown that minimizing the makespan is equivalent to minimizing $D(\sigma) = A_{\sigma(1)} + \sum_{k=1}^{n-1} d_{\sigma(k)\sigma(k+1)} + B_{\sigma(n)}$. Furthermore, this is equivalent to the minimization of $\sum_{k=1}^{n-1} d_{\sigma(k)\sigma(k+1)} + B_{\sigma(n)}$ *if* $\sigma(1)$, the first part to be processed, is fixed.

Consider an n-city TSP with distance matrix $C = (c_{ij}) = \min\{B_i + A''_j, \max\{\mu, B_i, A''_j\}\}$, where for a fixed $r \in \{1, 2, \ldots, n\}$ we have $A''_r = 0$, and $A''_j = A_j$ for $j \in \{1, 2, \ldots, n\} - r$. Since $A''_r = 0$, $c_{\sigma(n)r} = B_{\sigma(n)}$, and the length of the tour $(\sigma(1) = r, \sigma(2), \ldots, \sigma(n))$ is equal to $\sum_{k=1}^{n-1} c_{\sigma(k)\sigma(k+1)} + c_{\sigma(n)r} = \sum_{k=1}^{n-1} d_{\sigma(k)\sigma(k+1)} + B_{\sigma(n)}$. This implies that the minimum makespan of a schedule with a fixed first part can be found by solving a TSP.

Procedure Makespan-RRC2(r)
1. Let $A''_j = A_j$, $j \in \{1, 2, \ldots, n\} - r$.
2. Find the best tour $\sigma^*(r)$ of an n-city TSP with distance matrix $C = (c_{ij})$ $= \min\{B_i + A''_j, \max\{\mu, B_i, A''_j\}\}$ and $A''_r = 0$. Let $L^*(r)$ be its length.

Return: The best tour $\sigma^*(r)$ and $C^*_{\max}(r) = L^*(r) + A_r + \sum_{i=1}^{n} \Gamma(i) + n\Delta - 3\delta$.

Procedure Makespan-RRC2(r) contains the detailed steps of finding this minimum makespan. The best tour $\sigma^*(r)$ and its length $L^*(r)$ can be found in $O(n \log n)$ time (cf. Aneja and Kamoun (1999) and Section 5.3.2). Thus Procedure Makespan-RRC2(r) takes $O(n \log n)$ time.

Now by calling Procedure Makespan-RRC2(r) repeatedly for $r = 1, 2, \ldots, n$, which will take $O(n^2 \log n)$ time, we can find the optimal solution for the $RRC2|\ell = 2K|C_{\max}$ problem. The minimum makespan is $C^*_{\max} = \min_{r \in \{1,2,\ldots,n\}} C^*_{\max}(r)$. Therefore, we have proved the following theorem.

Theorem 2.7 *Problem $RRC2|\ell = 2K|C_{\max}$ can be solved optimally in $O(n^2 \log n)$ time.*

2.4 Three-machine Loop-reentrant Cells

There exist many different reentrant processing routes in an m-machine reentrant robotic cell, when $m \geq 3$. In this section, we examine a special class of reentrant processing where parts are to be processed by each machine as in a flow shop and then go back to the first machine for a finishing operation, i.e., the part processing route is $M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_m \rightarrow M_1$. As the part processing route behaves like a loop, this is referred to as *loop-reentrant processing* in Middendorf and Timkovsky (2002). Accordingly, we call such robotic cells *loop-reentrant robotic cells.* Loop-reentrant processing can frequently be encountered in real life applications. It reflects certain manufacturing environments in which there is a primary machine that is responsible for both pre- and post-processing, e.g., a cleaning procedure, while the rest of operations are to be performed on several secondary machines. Another example is electronic signal processing, in which signal pulses have to go to a computer for preprocessing, and then through the sensing and command system for transmission and retrieval, and finally back to the computer for postprocessing. Note that loop-reentrant processing has also been studied by Wang, Sethi, and van de Velde (1997), where it was called chain-reentrant processing. We add *loop* into the β field of the $\alpha|\beta|\gamma$ notation for loop-reentrant robotic-cell problems. Therefore, the problem of minimizing cycle time in an m-machine loop-reentrant robotic cell is identified by the three-tuple $RRCm|loop|C_t$.

The m-machine loop-reentrant robotic cell is, in a certain sense, a generalization of the regular m-machine robotic cell without reentry. In the regular m-machine robotic cell, the problem of finding the sequence of robot move cycles and the part processing sequence which jointly minimize the cycle time is strongly $\mathcal{NP}$-hard for $m \geq 3$ (Hall, Kamoun, and Sriskandarajah 1998). The strong $\mathcal{NP}$-hardness of $RRCm|loop|C_t$, where $m \geq 3$, however, does not follow from this in an obvious way. For example, the regular three-machine robotic cell has six potentially optimal robot move cycles for producing one unit, while a three-machine loop-reentrant

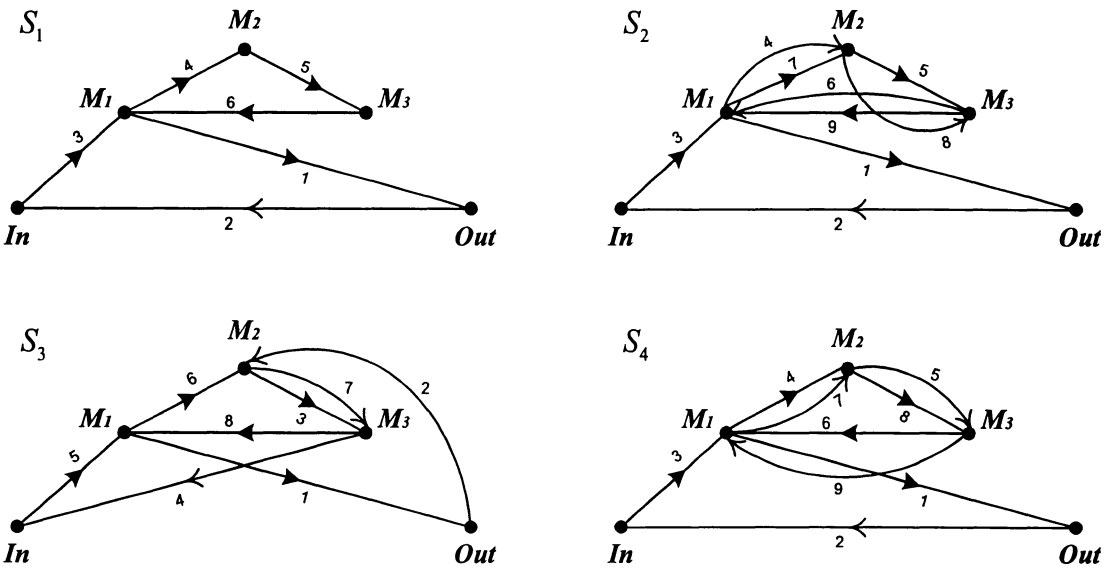

Figure 2.5: Robot move cycles in a three-machine loop-reentrant cell

robotic cell has only four potentially optimal cycles that produce one unit. To examine these cycles in detail, let us consider the system in state $\mathcal{G}$ when the robot is ready to unload a part from machine M_1 after this part has completed all operations, i.e., $\mathcal{G} = (\Omega, \cdot, \cdot, M_1^{(2)+})$. Beginning from state $\mathcal{G}$, the four robot move cycles may be described by a series of state transitions as follows. (Refer to Figure 2.5 for the steps of each cycle. The numbers on the arcs depict the order of robot moves.)

S_1: $(\Omega, \varnothing, \varnothing, M_1^{(2)+}), (\varnothing, \varnothing, \varnothing, O), (\varnothing, \varnothing, \varnothing, I), (\Omega, \varnothing, \varnothing, M_1^{(1)-}), (\Omega, \varnothing, \varnothing, M_1^{(1)+}),$
$(\varnothing, \Omega, \varnothing, M_2^-), (\varnothing, \Omega, \varnothing, M_2^+), (\varnothing, \varnothing, \Omega, M_3^-), (\varnothing, \varnothing, \Omega, M_3^+), (\Omega, \varnothing, \varnothing, M_1^{(2)-}),$
$(\Omega, \varnothing, \varnothing, M_1^{(2)+}),$

S_2: $(\Omega, \Omega, \varnothing, M_1^{(2)+}), (\varnothing, \Omega, \varnothing, O), (\varnothing, \Omega, \varnothing, I), (\Omega, \Omega, \varnothing, M_1^{(1)-}), (\Omega, \Omega, \varnothing, M_2^+),$
$(\Omega, \varnothing, \Omega, M_3^-), (\Omega, \varnothing, \Omega, M_1^{(1)+}), (\varnothing, \Omega, \Omega, M_2^-), (\varnothing, \Omega, \Omega, M_3^+), (\Omega, \Omega, \varnothing, M_1^{(2)-}),$
$(\Omega, \Omega, \varnothing, M_1^{(2)+}),$

S_3: $(\Omega, \Omega, \varnothing, M_1^{(2)+}), (\varnothing, \Omega, \varnothing, O), (\varnothing, \Omega, \varnothing, M_2^+), (\varnothing, \varnothing, \Omega, M_3^-), (\varnothing, \varnothing, \Omega, I),$
$(\Omega, \varnothing, \Omega, M_1^{(1)-}), (\Omega, \varnothing, \Omega, M_1^{(1)+}), (\varnothing, \Omega, \Omega, M_2^-), (\varnothing, \Omega, \Omega, M_3^+), (\Omega, \Omega, \varnothing, M_1^{(2)-}),$
$(\Omega, \Omega, \varnothing, M_1^{(2)+}),$

S_4: $(\Omega, \varnothing, \Omega, M_1^{(2)+}), (\varnothing, \varnothing, \Omega, O), (\varnothing, \varnothing, \Omega, I), (\Omega, \varnothing, \Omega, M_1^{(1)-}), (\Omega, \varnothing, \Omega, M_1^{(1)+}),$
$(\varnothing, \Omega, \Omega, M_2^-), (\varnothing, \Omega, \Omega, M_3^+), (\Omega, \Omega, \varnothing, M_1^{(2)-}), (\Omega, \Omega, \varnothing, M_2^+), (\Omega, \varnothing, \Omega, M_3^-),$
$(\Omega, \varnothing, \Omega, M_1^{(2)+}).$

2.4.1 Single cycles

We can derive the cycle time for the production of an MPS in sequence σ under each robot move cycle. Since we are dealing with cyclic production of the MPS, we always have $\sigma(n+k) = \sigma(k)$ for $k = 1, 2, ..., n$.

The elapsed time between the loading of parts $P_{\sigma(k)}$ and $P_{\sigma(k+1)}$ on machine M_1 for the first time under cycle S_1 is given by

$$\begin{aligned} T^1_{\sigma(k)\sigma(k+1)} &= a_{\sigma(k)1} + \varepsilon + \delta + \varepsilon + b_{\sigma(k)} + \varepsilon + \delta + \varepsilon + c_{\sigma(k)} + \varepsilon + 2\delta + \varepsilon + a_{\sigma(k)2} + \varepsilon \\ &\quad + 3\delta + \varepsilon + 4\delta + \varepsilon + \delta + \varepsilon \\ &= 10\varepsilon + 12\delta + a_{\sigma(k)1} + a_{\sigma(k)2} + b_{\sigma(k)} + c_{\sigma(k)}. \end{aligned}$$

Thus it is easy to see that, under cycle S_1, the production of an MPS in any sequence has the *same* cycle time $T_1 = n(10\varepsilon + 12\delta) + \sum_{i=1}^{n}(a_{i1} + a_{i2} + b_i + c_i)$. Therefore, *any* part processing sequence is trivially optimal for the $RRC3|loop, S_1|C_t$ problem.

Under cycle S_2, the time between the loading of part $P_{\sigma(k+1)}$ on machine M_2, and the loading of part $P_{\sigma(k+2)}$ on machine M_2, can be obtained as

$$\begin{aligned} T^2_{\sigma(k+1)\sigma(k+2)} &= \delta + w_3(k) + \varepsilon + 2\delta + \varepsilon + a_{\sigma(k)2} + \varepsilon + 3\delta + \varepsilon + 4\delta + \varepsilon + \delta + \varepsilon + \delta \\ &\quad + w_2(k+1) + \varepsilon + \delta + \varepsilon + 2\delta + w_1^{(1)}(k+2) + \varepsilon + \delta + \varepsilon \\ &= 10\varepsilon + 16\delta + a_{\sigma(k)2} + w_1^{(1)}(k+2) + w_2(k+1) + w_3(k), \end{aligned}$$

where

$$\begin{aligned} w_1^{(1)}(k+2) &= \max\{0,\ a_{\sigma(k+2)1} - (\delta + w_2(k+1) + \varepsilon + \delta + \varepsilon + 2\delta)\} \\ &= \max\{0,\ a_{\sigma(k+2)1} - 2\varepsilon - 4\delta - w_2(k+1)\} \end{aligned}$$

is the robot waiting time at M_1 for the 1st processing of part $P_{\sigma(k+2)}$,

$$w_2(k+1) = \max\{0,\ b_{\sigma(k+1)} - a_{\sigma(k)2} - 6\varepsilon - 12\delta - w_3(k)\}$$

is the robot waiting time at M_2 for part $P_{\sigma(k+1)}$, and

$$\begin{aligned} w_3(k) &= \max\{0,\ c_{\sigma(k)} - (2\delta + w_1^{(1)}(k+1) + \varepsilon + \delta + \varepsilon + \delta)\} \\ &= \max\{0,\ c_{\sigma(k)} - 2\varepsilon - 4\delta - w_1^{(1)}(k+1)\} \end{aligned}$$

is the robot waiting time at M_3 for part $P_{\sigma(k)}$.

Adding $w_1^{(1)}(k+2)$ and $w_2(k+1)$ and substituting for $w_2(k+1)$, $T^2_{\sigma(k+1)\sigma(k+2)}$ may be simplified as

$$\begin{aligned} T^2_{\sigma(k+1)\sigma(k+2)} &= 10\varepsilon + 16\delta + a_{\sigma(k)2} + \max\{w_2(k+1),\ a_{\sigma(k+2)1} - 2\varepsilon - 4\delta\} + w_3(k) \\ &= 4\varepsilon + 4\delta + \max\{4\varepsilon + 8\delta,\ 2\varepsilon + 4\delta + a_{\sigma(k+2)1},\ b_{\sigma(k+1)} - a_{\sigma(k)2} - 2\varepsilon \\ &\quad - 4\delta - w_3(k)\} + a_{\sigma(k)2} + 2\varepsilon + 4\delta + w_3(k), \end{aligned}$$

where

$$\begin{aligned} w_3(k) &= \max\{0,\ c_{\sigma(k)} - 2\varepsilon - 4\delta - \max\{0,\ a_{\sigma(k+1)1} - 2\varepsilon - 4\delta - w_2(k)\}\} \\ &= \max\{0,\ c_{\sigma(k)} - \max\{2\varepsilon + 4\delta,\ a_{\sigma(k+1)1} - \max\{0,\ b_{\sigma(k)} - a_{\sigma(k-1)2} - 6\varepsilon - 12\delta \\ &\quad - w_3(k-1)\}\}\}. \end{aligned}$$

Given a part processing sequence σ, the time to produce an MPS under cycle S_2 is then equal to

$$\begin{aligned} T_2(\sigma) &= \sum_{k=1}^{n} T^2_{\sigma(k+1)\sigma(k+2)} \\ &= n(4\varepsilon + 4\delta) + \sum_{k=1}^{n} \max\{4\varepsilon + 8\delta,\ 2\varepsilon + 4\delta + a_{\sigma(k+2)1},\ b_{\sigma(k+1)} - a_{\sigma(k)2} \\ &\quad - 2\varepsilon - 4\delta - w_3(k)\} + \sum_{k=1}^{n}(a_{\sigma(k)2} + 2\varepsilon + 4\delta) + \sum_{k=1}^{n} w_3(k), \end{aligned} \tag{2.6}$$

where $w_3(k) = \max\{0,\ c_{\sigma(k)} - \max\{2\varepsilon + 4\delta,\ a_{\sigma(k+1)1} - \max\{0,\ b_{\sigma(k)} - a_{\sigma(k-1)2} - 6\varepsilon - 12\delta - w_3(k-1)\}\}\}$.

Under cycle S_3, the time between the loading of part $P_{\sigma(k)}$ on machine M_3, and the loading of part $P_{\sigma(k+1)}$ on machine M_3, can be obtained as

$$\begin{aligned} T^3_{\sigma(k)\sigma(k+1)} &= 3\delta + \varepsilon + \delta + \varepsilon + a_{\sigma(k+1)1} + \varepsilon + \delta + \varepsilon + \delta + w_3(k) + \varepsilon + 2\delta + \varepsilon + a_{\sigma(k)2} \\ &\quad + \varepsilon + 3\delta + \varepsilon + 2\delta + w_2(k+1) + \varepsilon + \delta + \varepsilon \\ &= 10\varepsilon + 14\delta + a_{\sigma(k+1)1} + a_{\sigma(k)2} + w_2(k+1) + w_3(k), \end{aligned}$$

where

$$\begin{aligned} w_2(k+1) &= \max\{0,\ b_{\sigma(k+1)} - (\delta + w_3(k) + \varepsilon + 2\delta + \varepsilon + a_{\sigma(k)2} + \varepsilon + 3\delta + \varepsilon + 2\delta)\} \\ &= \max\{0,\ b_{\sigma(k+1)} - a_{\sigma(k)2} - 4\varepsilon - 8\delta - w_3(k)\} \end{aligned}$$

is the robot waiting time at M_2 for part $P_{\sigma(k+1)}$, and

$$\begin{aligned} w_3(k) &= \max\{0,\ c_{\sigma(k)} - (3\delta + \varepsilon + \delta + \varepsilon + a_{\sigma(k+1)1} + \varepsilon + \delta + \varepsilon + \delta)\} \\ &= \max\{0,\ c_{\sigma(k)} - a_{\sigma(k+1)1} - 4\varepsilon - 6\delta\} \end{aligned}$$

is the robot waiting time at M_3 for part $P_{\sigma(k)}$.

Adding $w_2(k+1)$ and $w_3(k)$, $T^3_{\sigma(k)\sigma(k+1)}$ may be simplified as

$$\begin{aligned} T^3_{\sigma(k)\sigma(k+1)} &= 10\varepsilon + 14\delta + a_{\sigma(k+1)1} + a_{\sigma(k)2} + \max\{w_3(k),\ b_{\sigma(k+1)} - a_{\sigma(k)2} - 4\varepsilon - 8\delta\} \\ &= 10\varepsilon + 14\delta + a_{\sigma(k+1)1} + a_{\sigma(k)2} + \max\{0,\ c_{\sigma(k)} - a_{\sigma(k+1)1} - 4\varepsilon - 6\delta, \\ &\quad b_{\sigma(k+1)} - a_{\sigma(k)2} - 4\varepsilon - 8\delta\}. \end{aligned}$$

Given a part processing sequence σ, the time to produce an MPS under cycle S_3 is then equal to

$$\begin{aligned} T_3(\sigma) &= \sum_{k=1}^{n} T^3_{\sigma(k)\sigma(k+1)} \\ &= n(10\varepsilon + 14\delta) + \sum_{k=1}^{n}(a_{\sigma(k+1)1} + a_{\sigma(k)2}) \\ &\quad + \sum_{k=1}^{n} \max\{0,\ c_{\sigma(k)} - a_{\sigma(k+1)1} - 4\varepsilon - 6\delta,\ b_{\sigma(k+1)} - a_{\sigma(k)2} - 4\varepsilon - 8\delta\}. \quad (2.7) \end{aligned}$$

Under cycle S_4, the time between the loading of part $P_{\sigma(k+1)}$ on machine M_3, and the loading of part $P_{\sigma(k+2)}$ on machine M_3, can be obtained as

$$\begin{aligned} T^4_{\sigma(k+1)\sigma(k+2)} &= 2\delta + w_1^{(2)}(k) + \varepsilon + 3\delta + \varepsilon + 4\delta + \varepsilon + \delta + \varepsilon + a_{\sigma(k+2)1} + \varepsilon + \delta + \varepsilon + \delta \\ &\quad + w_3(k+1) + \varepsilon + 2\delta + \varepsilon + \delta + w_2(k+2) + \varepsilon + \delta + \varepsilon \\ &= 10\varepsilon + 16\delta + a_{\sigma(k+2)1} + w_2(k+2) + w_3(k+1) + w_1^{(2)}(k), \end{aligned}$$

where

$$\begin{aligned} w_2(k+2) &= \max\{0,\ b_{\sigma(k+2)} - (\delta + w_3(k+1) + \varepsilon + 2\delta + \varepsilon + \delta)\} \\ &= \max\{0,\ b_{\sigma(k+2)} - 2\varepsilon - 4\delta - w_3(k+1)\} \end{aligned}$$

is the robot waiting time at M_2 for part $P_{\sigma(k+2)}$,

$$w_3(k+1) = \max\{0,\ c_{\sigma(k+1)} - a_{\sigma(k+2)1} - 6\varepsilon - 12\delta - w_1^{(2)}(k)\}$$

is the robot waiting time at M_3 for part $P_{\sigma(k+1)}$, and

$$\begin{aligned} w_1^{(2)}(k) &= \max\{0,\ a_{\sigma(k)2} - (\delta + w_2(k+1) + \varepsilon + \delta + \varepsilon + 2\delta)\} \\ &= \max\{0,\ a_{\sigma(k)2} - 2\varepsilon - 4\delta - w_2(k+1)\} \end{aligned}$$

is the robot waiting time at M_1 for the 2nd processing of part $P_{\sigma(k)}$.

Adding $w_2(k+2)$ and $w_3(k+1)$ and substituting $w_3(k+1)$, $T^4_{\sigma(k+1)\sigma(k+2)}$ may be simplified as

$$\begin{aligned} T^4_{\sigma(k+1)\sigma(k+2)} &= 10\varepsilon + 16\delta + a_{\sigma(k+2)1} + \max\{w_3(k+1),\ b_{\sigma(k+2)} - 2\varepsilon - 4\delta\} + w_1^{(2)}(k) \\ &= 4\varepsilon + 4\delta + \max\{4\varepsilon + 8\delta,\ c_{\sigma(k+1)} - a_{\sigma(k+2)1} - 2\varepsilon - 4\delta - w_1^{(2)}(k), \\ &\quad 2\varepsilon + 4\delta + b_{\sigma(k+2)}\} + a_{\sigma(k+2)1} + 2\varepsilon + 4\delta + w_1^{(2)}(k), \end{aligned}$$

where

$$\begin{aligned} w_1^{(2)}(k) &= \max\{0,\ a_{\sigma(k)2} - 2\varepsilon - 4\delta - \max\{0,\ b_{\sigma(k+1)} - 2\varepsilon - 4\delta - w_3(k)\}\} \\ &= \max\{0,\ a_{\sigma(k)2} - \max\{2\varepsilon + 4\delta,\ b_{\sigma(k+1)} - \max\{0,\ c_{\sigma(k)} - a_{\sigma(k+1)1} \\ &\quad - 6\varepsilon - 12\delta - w_1^{(2)}(k-1)\}\}\}. \end{aligned}$$

Given a part processing sequence σ, the time to produce an MPS under cycle S_4 is then equal to

$$\begin{aligned} T_4(\sigma) &= \sum_{k=1}^{n} T^4_{\sigma(k+1)\sigma(k+2)} \\ &= n(4\varepsilon + 4\delta) + \sum_{k=1}^{n} \max\{4\varepsilon + 8\delta,\ 2\varepsilon + 4\delta + b_{\sigma(k+2)},\ c_{\sigma(k+1)} - a_{\sigma(k+2)1} \\ &\quad - 2\varepsilon - 4\delta - w_1^{(2)}(k)\} + \sum_{k=1}^{n}(a_{\sigma(k+2)1} + 2\varepsilon + 4\delta) + \sum_{k=1}^{n} w_1^{(2)}(k), \end{aligned} \tag{2.8}$$

where $w_1^{(2)}(k) = \max\{0,\ a_{\sigma(k)2} - \max\{2\varepsilon + 4\delta,\ b_{\sigma(k+1)} - \max\{0,\ c_{\sigma(k)} - a_{\sigma(k+1)1} - 6\varepsilon - 12\delta - w_1^{(2)}(k-1)\}\}\}$.

Next we show that the problem of finding the best part processing sequence using robot move cycle S_i, i.e., $RRC3|loop, S_i|C_t$, is strongly $\mathcal{NP}$-hard for $i = 2, 3, 4$. We begin with the $i = 2$ case.

Observe the substantial similarity between the S_2 cycle and the robot move cycle S_2' in the regular three-machine robotic cell (Sethi et al. 1992). See Figure 2.6 for the S_2' cycle. Hall, Kamoun, and Sriskandarajah (1998) have derived that the cycle

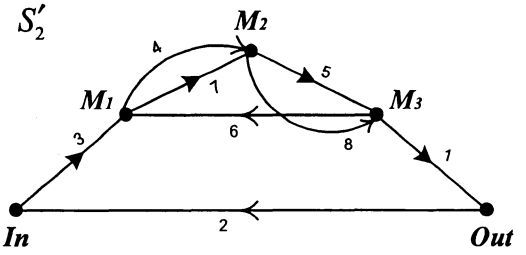

Figure 2.6: Cycle S_2' in the regular three-machine robotic cell

time for the production of an MPS in sequence σ under cycle S_2' is given by $T_2'(\sigma) = n(4\varepsilon+4\delta)+\sum_{k=1}^{n}\max\{4\varepsilon+8\delta,\ 2\varepsilon+4\delta+a'_{\sigma(k+2)},\ b'_{\sigma(k+1)}-w_3'(k)\}+\sum_{k=1}^{n} w_3'(k)$, where $w_3'(k) = \max\{0,\ c'_{\sigma(k)} - \max\{2\varepsilon + 4\delta,\ a'_{\sigma(k+1)} - \max\{0,\ b'_{\sigma(k)} - 4\varepsilon - 8\delta - w_3'(k-1)\}\}\}$, and a'_j, b'_j and c'_j represent the processing times of part j on machines M_1, M_2 and M_3, respectively, for $j = 1, 2, \ldots, n$. Moreover, they have shown that the problem of

determining the best part processing sequence to minimize $T_2'(\sigma)$ is strongly $\mathcal{NP}$-hard, i.e., the problem $RC3|S_2'|C_t$ is $\mathcal{NP}$-hard in the strong sense.

Consider our problem, $RRC3|loop, S_2|C_t$. Equation (2.6) shows how to compute the cycle time to produce an MPS in sequence σ under cycle S_2. By setting $a_{\sigma(k)1} = a'_{\sigma(k)}$, $b_{\sigma(k)} = b'_{\sigma(k)} + a_{\sigma(k-1)2} + 2\varepsilon + 4\delta$, and $c_{\sigma(k)} = c'_{\sigma(k)}$, it is easy to see that minimizing $T_2'(\sigma)$ is equivalent to minimizing $T_2(\sigma)$ because the term $\sum_{k=1}^{n}(a_{\sigma(k)2} + 2\varepsilon + 4\delta)$ in $T_2(\sigma)$ is independent of the part processing sequence. Furthermore, it can be verified that $w_3(k) = w_3'(k)$ always, as they are calculated by the same recursion. We have thus proved the following theorem.

Theorem 2.8 *$RRC3|loop, S_2|C_t$ is strongly $\mathcal{NP}$-hard.*

For $RRC3|loop, S_4|C_t$, Equation (2.8) shows how to compute the cycle time for an MPS in sequence σ under cycle S_4. Similarly to the above, if we set $b_{\sigma(k)} = a'_{\sigma(k)}$, $c_{\sigma(k+1)} = b'_{\sigma(k+1)} + a_{\sigma(k+2)1} + 2\varepsilon + 4\delta$, and $a_{\sigma(k)2} = c'_{\sigma(k)}$, then minimizing $T_2'(\sigma)$ is also equivalent to minimizing $T_4(\sigma)$. Note that the term $\sum_{k=1}^{n}(a_{\sigma(k+2)1} + 2\varepsilon + 4\delta)$ in $T_4(\sigma)$ is independent of part processing sequence. We have therefore proved the following theorem.

Theorem 2.9 *$RRC3|loop, S_4|C_t$ is strongly $\mathcal{NP}$-hard.*

For $RRC3|loop, S_3|C_t$, the problem is to find the best part processing sequence that minimizes $T_3(\sigma)$ defined in (2.7). Let $\overline{a}_{\sigma(k+1)} = a_{\sigma(k+1)1} + 4\varepsilon + 6\delta$ and $\overline{d}_{\sigma(k)} = a_{\sigma(k)2} + 4\varepsilon + 8\delta$. We can rewrite $T_3(\sigma)$ as $T_3(\sigma) = 2n\varepsilon + \sum_{k=1}^{n}(\overline{a}_{\sigma(k+1)} + \overline{d}_{\sigma(k)}) + \sum_{k=1}^{n}\max\{0, c_{\sigma(k)} - \overline{a}_{\sigma(k+1)}, b_{\sigma(k+1)} - \overline{d}_{\sigma(k)}\}$. Sriskandarajah, Hall, and Kamoun (1998) have studied part sequencing problems in a regular four-machine robotic cell. They have shown that the problem of finding the best part processing sequence that minimizes $T''(\sigma) = \sum_{k=1}^{n}(a''_{\sigma(k+1)} + d''_{\sigma(k)}) + \sum_{k=1}^{n}\max\{0, c''_{\sigma(k)} - a''_{\sigma(k+1)}, b''_{\sigma(k+1)} - d''_{\sigma(k)}\}$, where a''_j, b''_j, c''_j, and d''_j are the modified processing times of part j on machines M_1, M_2, M_3, and M_4, respectively, $j = 1, \ldots, n$, is strongly $\mathcal{NP}$-hard. (Refer to cycle $S_{9,4}$ in their paper.) If we let $c_{\sigma(k)} = c''_{\sigma(k)}$, $\overline{a}_{\sigma(k+1)} = a''_{\sigma(k+1)}$, $b_{\sigma(k+1)} = b''_{\sigma(k+1)}$, and $\overline{d}_{\sigma(k)} = d''_{\sigma(k)}$, then it is clear that minimizing $T''(\sigma)$ is equivalent to minimizing $T_3(\sigma)$. Hence, we have proved the following theorem.

Theorem 2.10 *$RRC3|loop, S_3|C_t$ is strongly $\mathcal{NP}$-hard.*

2.4.2 The general problem

In this section, we study the general problem of finding the sequence of 1-unit robot move cycles and the part processing sequence which jointly minimize the cycle time in three-machine loop-reentrant robotic cells. Note that the combinations of cycles S_1, S_2, S_3, S_4 are allowed for the general problem.

Before we present our result, let us first examine the $RC3|S_2'|C_t$ problem in detail. Recall that, for $RC3|S_2'|C_t$, its strong $\mathcal{NP}$-hardness has been proved by a polynomial transformation of the *Numerical Matching with Target Sums* (NMTS) problem to the decision version of the problem (Hall, Kamoun, and Sriskandarajah 1998). The NMTS problem is known to be $\mathcal{NP}$-complete in the strong sense (Garey and Johnson 1979), and may be stated as follows:

NMTS

Instance: Given three sets of positive integers $\mathcal{X} = \{x_1, \ldots, x_t\}$, $\mathcal{Y} = \{y_1, \ldots, y_t\}$, and $\mathcal{Z} = \{z_1, \ldots, z_t\}$, where $\sum_{j=1}^{t} x_j = \sum_{j=1}^{t} y_j + \sum_{j=1}^{t} z_j = X$.

Question: Can $\mathcal{Y} \cup \mathcal{Z}$ be partitioned into t disjoint subsets $\mathcal{A}_1, \ldots, \mathcal{A}_t$, each containing exactly one element from each of $\mathcal{Y}$ and $\mathcal{Z}$, such that $\sum_{a_i \in \mathcal{A}_j} a_i = x_j$ for $j = 1, \ldots, t$?

For a given instance of NMTS, we can create an instance Q' of the $RC3|S_2'|C_t$ problem with the MPS consisting of three part types and $n' = 3t$ parts as follows:

Part set $\mathcal{P}' = \{P_j', 1 \le j \le t\} \cup \{P_j', t+1 \le j \le 2t\} \cup \{P_j', 2t+1 \le j \le 3t\}$. Let a_j', b_j', and c_j' denote the processing times of part j on machine M_1, M_2, and M_3, respectively, $j = 1, \ldots, 3t$.

$$
\begin{array}{llll}
a_j' = E + x_j - \beta/2, & b_j' = 3E, & c_j' = \beta/2, & j = 1, \ldots, t, \\
a_j' = 3E - \beta/2, & b_j' = 2E, & c_j' = X - y_{j-t} + \beta/2, & j = t+1, \ldots, 2t, \\
a_j' = 2E - \beta/2, & b_j' = E + X + z_{j-2t}, & c_j' = \beta/2, & j = 2t+1, \ldots, 3t,
\end{array}
$$

where $X = \sum_{j=1}^{t} x_j$, $E = 3tX$, $\varepsilon = \delta = X/4$, and $\beta = 4\varepsilon + 8\delta$. The threshold cycle time for the production of n' parts is $G' = (6t+2)E + tX + Z$, where $Z = \sum_{j=1}^{t} z_j$.

The proof of Theorem 11 in Hall, Kamoun, and Sriskandarajah (1998) can be used to show that to achieve the threshold cycle time G' for problem instance Q', the elapsed time between the loading of parts $P_{\sigma(k+1)}$ and $P_{\sigma(k+2)}$ on machine M_2 must be equal to $T'^2_{\sigma(k+1)\sigma(k+2)} = 4\varepsilon + 4\delta + a'_{\sigma(k+2)} + \beta/2 + w_3'(k)$, where $\sum_{k=1}^{3t} w_3'(k) = tX - Y$ and $Y = \sum_{j=1}^{t} y_j$. See Hall, Kamoun, and Sriskandarajah (1998) for details.

Now consider the problem $RRC3|loop, S_2|C_t$. As shown in the proof of Theorem 2.8, by setting $a_{\sigma(k)1} = a'_{\sigma(k)}$, $b_{\sigma(k)} = b'_{\sigma(k)} + a_{\sigma(k-1)2} + 2\varepsilon + 4\delta$, and $c_{\sigma(k)} = c'_{\sigma(k)}$, minimizing $T_2'(\sigma)$ is equivalent to minimizing $T_2(\sigma)$. Hence, we can create from Q' an instance Q of the $RRC3|loop, S_2|C_t$ problem with an MPS consisting of three part types and $n = 3t$ parts as follows:

Part set $\mathcal{P} = \{P_j, 1 \le j \le t\} \cup \{P_j, t+1 \le j \le 2t\} \cup \{P_j, 2t+1 \le j \le 3t\}$. Let a_{j1}, b_j, c_j, and a_{j2} denote the processing times of part j on machine M_1 (the 1st time), M_2, M_3, and M_1 (the 2nd time), respectively, $j = 1, \ldots, 3t$.

$$
\begin{array}{llll}
a_{j1} = E + x_j - \beta/2, & b_j = 3E + 2X, & c_j = \beta/2, & j = 1, \ldots, t, \\
a_{j1} = 3E - \beta/2, & b_j = 2E + 2X, & c_j = X - y_{j-t} + \beta/2, & j = t+1, \ldots, 2t, \\
a_{j1} = 2E - \beta/2, & b_j = E + 3X + z_{j-2t}, & c_j = \beta/2, & j = 2t+1, \ldots, 3t,
\end{array}
$$

and $a_{j2} = X/2$ for $j = 1, \ldots, 3t$, where $E = 3tX$, $\varepsilon = \delta = X/4$, and $\beta = 4\varepsilon + 8\delta$. Since $T_2(\sigma) - T_2'(\sigma) = \sum_{k=1}^{n}(a_{\sigma(k)2} + 2\varepsilon + 4\delta) = \sum_{k=1}^{3t}(X/2 + X/2 + X) = 6tX = 2E$, the threshold cycle time for the production of n parts in Q is $G = G' + 2E = (6t + 4)E + tX + Z$, where $Z = \sum_{j=1}^{t} z_j$. Furthermore, since $2E$ is a constant, minimizing $T_2(\sigma)$ for instance Q is equivalent to minimizing $T_2'(\sigma)$ for instance Q', and a part processing sequence σ attains the threshold cycle time G' for Q' if and only if it attains the threshold cycle time G for Q. As mentioned above, a part processing sequence σ achieves the cycle time G' for Q' if the elapsed time between the loading of parts $P_{\sigma(k+1)}$ and $P_{\sigma(k+2)}$ on machine M_2 is exactly $T'^2_{\sigma(k+1)\sigma(k+2)} = 4\varepsilon + 4\delta + a'_{\sigma(k+2)} + \beta/2 + w_3'(k)$ for every k, where $w_3'(k) \leq \max\{0, c'_{\sigma(k)} - \beta/2\} < X$. For instance Q, this means that the elapsed time between the loading of parts $P_{\sigma(k+1)}$ and $P_{\sigma(k+2)}$ on machine M_2 must be equal to $T^2_{\sigma(k+1)\sigma(k+2)} = 4\varepsilon + 4\delta + (a_{\sigma(k+2)1} + \beta/2) + (a_{\sigma(k)2} + 2\varepsilon + 4\delta) + w_3(k) = 8\varepsilon + 12\delta + a_{\sigma(k+2)1} + a_{\sigma(k)2} + w_3(k)$, where $w_3(k) \leq \max\{0, c_{\sigma(k)} - \beta/2\} < X$ for each k. Since $w_3(k) < X$ and $a_{j2} = X/2$ for $j = 1, \ldots, 3t$, $\varphi \doteq T^2_{\sigma(k+1)\sigma(k+2)} = 8\varepsilon + 12\delta + a_{\sigma(k+2)1} + a_{\sigma(k)2} + w_3(k) < 8\varepsilon + 12\delta + a_{\sigma(k+2)1} + X/2 + X = 13X/2 + a_{\sigma(k+2)1}$. We show next that the times of other robot move cycles are strictly larger than the quantity φ.

Under cycle S_1, the robot needs to wait for part $P_{\sigma(k+1)}$ finishing its processing on M_2 and M_3, as well as the second operation on M_1, then wait for the next part, $P_{\sigma(k+2)}$, finishing its processing on M_1 before it is loaded on M_2, which take at least $b_{\sigma(k+1)} + c_{\sigma(k+1)} + a_{\sigma(k+1)2} + a_{\sigma(k+2)1} > \varphi$ units of time.

Under cycle S_3, the robot performs at least the following activities: move to M_3:(δ), unload part $P_{\sigma(k)}$ from M_3:(ε), move it back to M_1:(2δ) and load the part:(ε), wait for its processing:($a_{\sigma(k)2}$), unload the part:(ε), move it to *Out*:(3δ) and drop the part:(ε), move to M_2:(2δ), unload part $P_{\sigma(k+1)}$ from M_2:(ε), move it to M_3:(δ) and load the part:(ε), move to *In*:(3δ), pick up the next part, $P_{\sigma(k+2)}$:(ε), move it to M_1:(δ) and load the part:(ε), wait for its processing:($a_{\sigma(k+2)1}$), unload the part:(ε), move it to M_2:(δ) and load the part:(ε). These activities, not counting the possible waiting times, take $10\varepsilon + 14\delta + a_{\sigma(k)2} + a_{\sigma(k+2)1} = 13X/2 + a_{\sigma(k+2)1} > \varphi$ units of time.

Under cycle S_4, the robot performs at least the following activities: move to M_3:(δ), unload part $P_{\sigma(k)}$ from M_3:(ε), move it back to M_1:(2δ) and load the part:(ε), move to M_2:(δ), unload part $P_{\sigma(k+1)}$ from M_2:(ε), move it to M_3:(δ) and load the part:(ε), move to M_1:(2δ), unload part $P_{\sigma(k)}$ from M_1:(ε), move it to *Out*:(3δ) and drop the part:(ε), move to *In*:(4δ), pick up the next part, $P_{\sigma(k+2)}$:(ε), move it to M_1:(δ) and load the part:(ε), wait for its processing:($a_{\sigma(k+2)1}$), unload the part:(ε), move it to M_2:(δ) and load the part:(ε). These activities, not counting the possible waiting times, take $10\varepsilon + 16\delta + a_{\sigma(k+2)1} = 13X/2 + a_{\sigma(k+2)1} > \varphi$ units of time.

The above argument shows that the cycle times for the production of an MPS for instance Q under robot move cycles other than S_2 are larger than G, and switching to other cycles will increase the cycle time. Thus, S_2 is the only robot move cycle that can achieve the threshold cycle time of G for instance Q. Since there exists a schedule

Table 2.3: Complexity Results for Scheduling in Reentrant Robotic Cells

Problem	Complexity	Reference
$RRC2\|\ell = 2K\|C_t,\ K \geq 1$	$O(n \log n)$	Corollary 2.4
$RRC2\|\ell = 2K\|C_{\max},\ K \geq 1$	$O(n^2 \log n)$	Theorem 2.7
$RRC2\|\ell = 2K+1\|C_t,\ K \geq 1$	trivial	Section 2.3.1
$RRC2\|\ell = 2K+1\|C_{\max},\ K \geq 1$	trivial	Section 2.3.1
$RRC3\|loop, S_1\|C_t$	trivial	Section 2.4.1
$RRC3\|loop, S_i\|C_t,\ i = 2,3,4$	strongly $\mathcal{NP}$-hard	Theorem 2.8, 2.10, 2.9
$RRCm\|loop\|C_t,\ m \geq 3$	strongly $\mathcal{NP}$-hard	Theorem 2.11

σ for $RC3|S_2'|C_t$ on problem instance Q' with cycle time $T_2'(\sigma) \leq G'$ if and only if there exists a solution to the NMTS problem (Hall, Kamoun, and Sriskandarajah 1998), this implies the following result.

Theorem 2.11 *The $RRC3|loop|C_t$ problem is strongly $\mathcal{NP}$-hard.*

2.5 Summary

We have studied the problem of scheduling to minimize the cycle time or makespan in m-machine reentrant robotic cells. The decisions to be made include determining the sequence of 1-unit robot move cycles and the part processing sequence. We have established some interesting new complexity results sharpening the boundary between easy and hard problems. When $m = 2$, the cycle time minimization problem can be formulated as a solvable special case of the TSP, and hence is polynomially solvable. We have also showed that the makespan minimization problem can be solved in $O(n^2 \log n)$ time. When $m = 3$, we studied a special class of reentrant robotic cells named loop-reentrant robotic cells with the cycle time objective. We showed that, under three out of four potentially optimal robot move cycles that produce one unit, the part sequencing problem is strongly $\mathcal{NP}$-hard. Under the remaining cycle, the part sequencing problem can be solved easily. Finally, the general problem without restriction to any robot move cycle was proved to be strongly $\mathcal{NP}$-hard too. A summary of our complexity results for scheduling in reentrant robotic cells appears in Table 2.3.

Chapter 3
Scheduling Multi-component Parts in a Robotic Cell

3.1 Introduction

3.1.1 The model

Consider a generalization of the classical robotic-cell scheduling model as follows: In a flexible manufacturing cell, there are two machines, M_1 and M_2, and a robot all without buffer. The parts are available at the input station (In) at time zero. Each part i ($i = 1, 2, ..., n$) consists of $K \geq 1$ identical components to be processed *together* first on machine M_1 for a_i time, then processed on machine M_2 *item-by-item*, each component requiring b_i processing time. As each part involves one processing step at the first machine and K processing steps at the second machine, we label this type of processing as 1-K *processing*.

For each part, the robot first picks it up at In, then moves it to the first machine, M_1, and loads it on that machine for processing; after the processing is completed, the robot unloads the part and moves it to the second machine, M_2, for processing its individual components. Since the components need the robot to perform the loading and unloading tasks at M_2 before or after each of the K processing steps, these K steps cannot be combined into one single step. After all of its components are processed on M_2, the robot moves the finished part to an output station (Out) and drops it there. During the entire process, both the operations of the robot and the processing of parts (components) on machines are nonpreemptive. The K components of a part are handled as a whole except at machine M_2. In, M_1, M_2, and Out are located on the arc of a circle or on a straight line with the robot at the center. Figure 3.1 exhibits the movement of a part (or component) in such a two-machine robotic cell. The input and output stations have unlimited storage capacity. Since there is no buffer space on machines, any part (component) being produced must be either on one of the machines or on the robot. Neither a machine nor the robot can handle more than one part at a time.

This model has a variety of applications. Our study was, in particular, inspired by a real-life application in an automated pharmaceutical laboratory in which a large

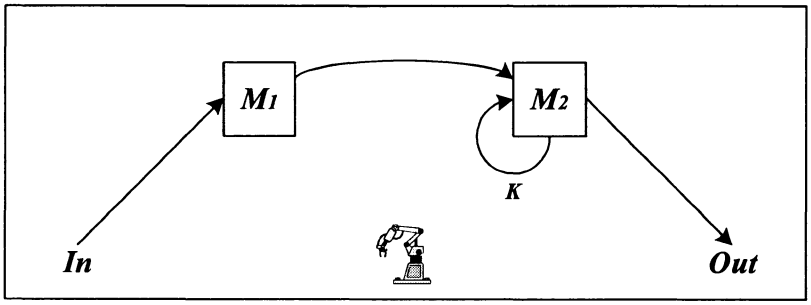

Figure 3.1: 1-K processing in a two-machine robotic cell

number of samples need to go through a given chemical process. Each sample is a set of test tubes. Sample i ($1 \leq i \leq n$) is composed of K test tubes. After a sample is picked up from an input station, it is moved to an instrument (M_1)—a versatile and flexible (multifunctional) machine—for a chemical process. When the process finishes, the sample is moved to the reader (M_2) to measure the data for each test tube (one by one). When all measurements are taken, the sample is dropped at an output station. To realize automation and increase efficiency, a robot performs all moving, loading and unloading tasks during the whole process.

This model is also encountered in semiconductor test facilities consisting of burn-in ovens and particular testers in sequence, where the oven is viewed as a batch processing machine and the tester is modeled as a unit-capacity machine. A batch processing machine is one where a number of components (jobs) are processed together as a batch. That is, similar tasks are batched or grouped together to avoid setup time or setup cost. Batch processing machines are widely used in flexible manufacturing. Examples of batch processing machines in semiconductor manufacturing are etchers in wafer fab and burn-in ovens in final test. Here the processing of each batch consists of two stages. The first stage is undertaken on the batch processing machine common to all components in a batch. All of these components start and finish processing at the same time. The second stage is undertaken on the unit-capacity machine for each component at a time. Components are regarded as single indivisible entities. Thus, once the processing of a component on a machine has started, this operation must be completed before this component can be processed on any other machines.

In this chapter, we concentrate on the scheduling of operations in such robotic cells. We examine both the problem of minimizing the maximum completion time (i.e., the *makespan*) for producing a given set of parts and the problem of minimizing the cycle time of a minimal part set (MPS) in cyclic production. (A formal definition of MPS and related issues are provided in Section 1.2.) Notice that there is only one robot in the normal setting. Thus the robot is often the bottleneck of systems. Therefore, unlike the traditional scheduling which only concerns the processing sequences of parts, both the robot move activities and the part processing sequences should be optimized in order to improve the productivity of these systems.

We use the following notation to describe the scheduling model.

n = the total number of parts to be produced or the number of parts in an MPS.
K = the number of identical components in a part, $K \geq 1$. It is a constant.
σ = the processing sequence of parts, $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$.
$P_{\sigma(k)}$ = the kth part in the sequence σ to be produced, $k = 1, 2, \ldots, n$.
a_i = the processing time of part i on (machine) M_1.
b_i = the processing time required for each component of part i on M_2.
δ = the travel time taken by the robot to move between adjacent location pairs (In, M_1), (M_1, M_2), and (M_2, Out). Travel between any two locations is via intermediate locations. Thus the movement, e.g., from In to Out takes 3δ time.
ε = the time needed to pick up, (un)load, or drop a part (or component) by the robot.

Throughout this chapter, all numerical data are assumed to be non-negative integers, unless explicitly denoted otherwise. For the sake of brevity, we assume that the travel, pick up, load, unload, and drop times are independent of the machines, but all results in this chapter are also applicable to machine-dependent travel times and (un)loading times.

Following the three-field notation $\alpha|\beta|\gamma$ for machine scheduling problems, we employ $RC2$ under α to indicate a two-machine robotic cell. Under β, we use 1-K to indicate the 1-K processing. Under γ, we may have $C_{\max}$ or C_t to denote the makespan or cycle time objective, respectively. For instance, $RC2|1\text{-}K|C_t$ means minimizing cycle time of producing K-component parts with 1-K processing in a two-machine robotic cell.

Remark The mirror image of 1-K processing is K-1 *processing* in which each part has K processing steps at M_1 and one processing step at M_2, see Figure 3.2. Many real life manufacturing processes follow this model. For example, in assembly of printed circuit boards (PCBs), several components should be inserted at specified points on a bare PCB by a so-called component placement machine before a postprocessing step, for instance, a cooling, inspection or packaging operation by another machine. Due to the reversibility of the process, it is easy to see that the results for 1-K processing are also applicable to K-1 processing.

3.1.2 Previous related work

There has been a lot of research work on different variants of robotic-cell scheduling. For traditional production with makespan objective, Kise, Shioyama, and Ibaraki (1991) study a two-machine no-buffer flow shop with part transportation done by an Automated Guided Vehicle (AGV). An $O(n^2 \log n)$ algorithm is given

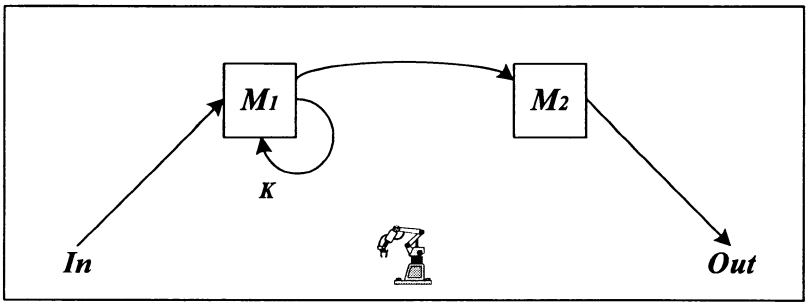

Figure 3.2: K-1 processing in a two-machine robotic cell

to find the optimal part sequence to minimize the makespan under a given AGV move cycle. When the machines have unlimited buffer, Kise (1991) proves the problem to be $\mathcal{NP}$-hard. More complexity results for special cases with unlimited buffer are identified by Hurink and Knust (2001). Agnetis, Pacciarelli, and Rossi (1996) analyze the lot scheduling in a two-machine cell where the machines are equipped with a swapping device, allowing the AGV to exchange a new part for the one released by a machine. Each lot is composed of a given number of identical parts to be processed consecutively. The goal is to determine a sequence of n lots so that the makespan is minimized. While the problem is $\mathcal{NP}$-hard, an optimal solution that runs in $O(n \log n)$ time is given for a special case. Panwalkar (1991) considers a two-machine flow shop in which parts are transported from the first machine to the second by a robot. While the first machine has no buffer, the second machine has unlimited buffer. Panwalkar presents an $O(n^2)$ algorithm to minimize the makespan. Levner, Kogan, and Levin (1995) extend Panwalkar's results by allowing non-negligible loading/unloading times, and suggest an $O(n \log n)$ algorithm for finding an optimal part sequence that minimizes the makespan. Levner, Kogan, and Maimon (1995) deal with a robotic cell with part dependent transportation and setup effects, and derive an $O(n^3)$ algorithm to minimize the makespan. Hertz, Mottet, and Rochat (1996) investigate a scheduling problem in a robotized analytical system where a chemical treatment has to be performed on each one of n identical samples. The chemical treatment is represented by a predetermined ordered set of tasks to be carried out on several resources. Here a robot is used to transport the samples between the resources (i.e., machines and storage units). The time spent by a sample in a resource is not fixed in advance but bounded by given minimal and maximal values. Moreover, it is imposed that the time between the end of a task and the beginning of another task be within a given maximal value. A heuristic algorithm is proposed to minimize the makespan.

For cyclic scheduling in robotic cells, Crama and van de Klundert (1997) present an $O(m^3)$ dynamic programming algorithm to find the optimal robot move sequence to minimize the cycle time in an m-machine robotic cell with identical parts. Sethi et al. (1992) explore a two-machine robotic cell and show that two robot move

cycles exist. For any given robot move cycle, they find the optimal part processing sequence to minimize the cycle time. Hall, Kamoun, and Sriskandarajah (1997) propose an $O(n^4)$ algorithm that optimizes both the robot move cycle and part processing sequence for multiple parts in the two machine case. Aneja and Kamoun (1999) improve the algorithm and present an $O(n \log n)$ algorithm based on the well-known Gilmore–Gomory algorithm (Gilmore and Gomory 1964) for a special case of the traveling salesman problem (TSP). Agnetis (2000) considers a similar problem with no-wait-in-process constraint and shows that the problem of minimizing cycle time is also solvable in $O(n \log n)$ time for multiple parts in two-machine robotic cells. For a general survey on robotic scheduling problems, we refer the reader to Crama et al. (2000) and Middendorf and Timkovsky (2002).

The remainder of the chapter is arranged as follows. In Section 3.2, we analyze the movement of the robot and establish that there are only two potentially optimal robot move cycles in the cell. In Section 3.3 we formulate the problem of minimizing cycle time as a TSP and show that it is solvable in $O(n \log n)$ time. Section 3.4 investigates the makespan version of the problem. Finally, Section 3.5 concludes this chapter.

3.2 Problem Analysis

A robot move cycle is a sequence of robot moves that returns to the initial state. A 1-unit cycle returns to the same state after the production of a single unit (part). Recognizing that a robotic cell is a discrete dynamic system, Sethi et al. (1992) have defined the robot move cycle as a sequence of discrete state transitions. We extend their representation to accommodate the 1-K processing in the two-machine robotic cell. More specifically, a state of the system is defined by a 3-tuple $\mathcal{G} = (g_1, g_2, g_3)$, where $g_i \in \{\varnothing, \Omega\}$, $i = 1, 2$, and $g_3 \in \{I, O, M_j^{(h)-}, M_j^{(h)+}\}$, $j = 1, 2$. Here $g_i = \varnothing$ or Ω means that machine M_i is not occupied or is occupied by a part (component), respectively; g_3 refers to the robot position with I denoting the robot at In just as it arrives there and begins to pick up a part, O denoting the robot just as it completes the dropping of a part at Out and begins to leave there, $M_j^{(h)-}$ denoting the robot at M_j just after it has finished loading the hth component of a part on the machine, $M_j^{(h)+}$ denoting the robot at M_j just before it begins unloading the hth component of a part from the machine. As all the components of a part are treated together at M_1, the h superscript is omitted if the part is processed on M_1.

Let us consider the system in the initial state $\mathcal{G}$ when the robot has just loaded the 1st component of a part on machine M_2, i.e., $\mathcal{G} = (\varnothing, \Omega, M_2^{(1)-})$. There are two types of feasible robot move cycles, S_1 and S_2 (shown in Figures 3.3 and 3.4), to continue from state $\mathcal{G}$: The robot can either wait for the current part to finish all of its processing on machine M_2 (S_1) or move to the input station to pick up a new part some time during the processing of the current part on machine M_2 (S_2). Described

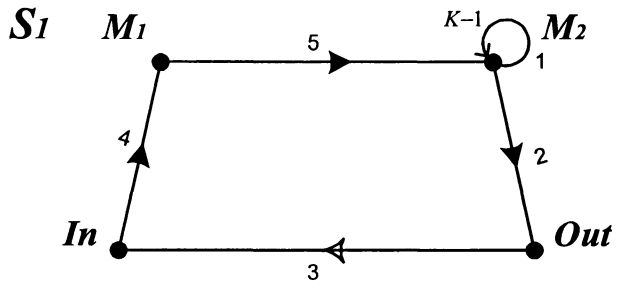

Figure 3.3: The robot move cycle S_1

by a sequence of states, the robot move cycles are

$$S_1\text{: } (\varnothing, \Omega, M_2^{(1)-}), (\varnothing, \Omega, M_2^{(1)+}), \underbrace{(\varnothing, \Omega, M_2^{(h)-}), (\varnothing, \Omega, M_2^{(h)+})}_{\text{repeat } K-1 \text{ times for } h=2,3,\ldots,K}, (\varnothing, \varnothing, O), (\varnothing, \varnothing, I),$$
$$(\Omega, \varnothing, M_1^-), (\Omega, \varnothing, M_1^+), (\varnothing, \Omega, M_2^{(1)-})$$

and

$$S_2^r\text{: } (\varnothing, \Omega, M_2^{(1)-}), \underbrace{(\varnothing, \Omega, M_2^{(h-1)+}), (\varnothing, \Omega, M_2^{(h)-})}_{\text{repeat } r-1 \text{ times for } h=2,3,\ldots,r}, (\varnothing, \Omega, I), (\Omega, \Omega, M_1^-), (\Omega, \Omega, M_2^{(r)+}),$$
$$\underbrace{(\Omega, \Omega, M_2^{(h)-}), (\Omega, \Omega, M_2^{(h)+})}_{\text{repeat } K-r \text{ times for } h=r+1,r+2,\ldots,K}, (\Omega, \varnothing, O), (\Omega, \varnothing, M_1^+), (\varnothing, \Omega, M_2^{(1)-})$$

for $r = 1, 2, \ldots, K$. Without loss of generality, let us assume that the current part is $P_{\sigma(k)}$. Then

the activities and their time requirements in robot move cycle S_1 ***are:*** Wait for the 1st component of part $P_{\sigma(k)}$ to finish its processing on M_2:$(b_{\sigma(k)})$, when it is finished, unload the component:(ε), after this, repeat $K-1$ times the following subcycle of activities in braces for $h = 2, 3, \ldots, K$ {load the hth component on M_2:(ε), wait for its processing:$(b_{\sigma(k)})$, unload it:(ε)}, then, move the part to *Out*:(δ), drop it there:(ε), move to *In*:(3δ), pick up the next part $P_{\sigma(k+1)}$:(ε), move it to M_1:(δ), and load it on M_1:(ε), wait for its processing:$(a_{\sigma(k+1)})$, unload the part:(ε), move it to M_2:(δ), and load the 1st component of part $P_{\sigma(k+1)}$ on M_2:(ε). (See Figure 3.3. The numbers $1, 2, \ldots$ on the arcs show the sequence of steps; $K-1$ represents the number of repeats.)

The time required by robot move cycle S_1 can be calculated as follows:

$$\begin{aligned} T_1 &= b_{\sigma(k)} + \varepsilon + (K-1)(\varepsilon + b_{\sigma(k)} + \varepsilon) + \delta + \varepsilon + 3\delta + \varepsilon + \delta + \varepsilon + a_{\sigma(k+1)} + \varepsilon + \delta + \varepsilon \\ &= 6\varepsilon + 6\delta + b_{\sigma(k)} + (K-1)(2\varepsilon + b_{\sigma(k)}) + a_{\sigma(k+1)}. \end{aligned} \tag{3.1}$$

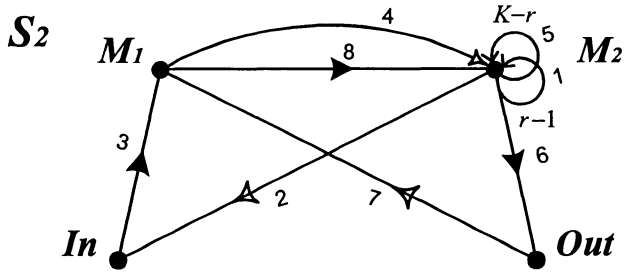

Figure 3.4: The robot move cycle S_2^r

Depending on the time when the robot moves from machine M_2 to the input station, the second type of robot move cycle has K possible scenarios. In the rth scenario, denoted by S_2^r $(r = 1, 2, \ldots, K)$, the robot moves over to the input station after loading the rth component of the current part on machine M_2.

The activities and their time requirements in robot move cycle S_2^r are: Repeat $r-1$ times the following subcycle of activities in braces for $h = 2, 3, \ldots, r$ {wait until the $(h-1)$th component of part $P_{\sigma(k)}$ finishes its processing on M_2:$(b_{\sigma(k)})$, unload the component:(ε), load the hth component on M_2:(ε)}, then, move to In:(2δ), pick up the next part $P_{\sigma(k+1)}$:(ε), move it to M_1:(δ), and load it on M_1:(ε), move to M_2:(δ), if needed, wait for the rth component of part $P_{\sigma(k)}$ to finish its processing on M_2:$(w_2(k))$, unload it:(ε), repeat $K-r$ times the following subcycle of activities in braces for $h = r+1, r+2, \ldots, K$ {load the the hth component of $P_{\sigma(k)}$ on M_2:(ε), wait for its processing:$(b_{\sigma(k)})$, unload it:(ε)}, then, move $P_{\sigma(k)}$ to Out:(δ), drop it there:(ε), move to M_1:(2δ), if needed, wait for the part $P_{\sigma(k+1)}$ to finish its processing on M_1:$(w_1^r(k+1))$, unload it:(ε), move it to M_2:(δ), and load the 1st component of part $P_{\sigma(k+1)}$ on M_2:(ε). (See Figure 3.4. The numbers $1, 2, \ldots$ on the arcs show the sequence of steps; $r-1$ and $K-r$ represent the number of repeats.)

The time required by robot move cycle S_2^r can be calculated as follows:

$$\begin{aligned}
T_2^r &= (r-1)(b_{\sigma(k)} + 2\varepsilon) + 2\delta + \varepsilon + \delta + \varepsilon + \delta + w_2(k) + \varepsilon + (K-r)(\varepsilon + b_{\sigma(k)} + \varepsilon) \\
&\quad + \delta + \varepsilon + 2\delta + w_1^r(k+1) + \varepsilon + \delta + \varepsilon \\
&= 4\varepsilon + 4\delta + (r-1)(2\varepsilon + b_{\sigma(k)}) + 2\varepsilon + 4\delta + w_2(k) + (K-r)(\varepsilon + b_{\sigma(k)} + \varepsilon) \\
&\quad + w_1^r(k+1) \\
&= 4\varepsilon + 4\delta + (r-1)(2\varepsilon + b_{\sigma(k)}) + \max\{w_2(k) + 2\varepsilon + 4\delta + (K-r)(2\varepsilon + b_{\sigma(k)}), \\
&\quad a_{\sigma(k+1)}\} \\
&= 4\varepsilon + 4\delta + (r-1)(2\varepsilon + b_{\sigma(k)}) + \max\{2\varepsilon + 4\delta + (K-r)(2\varepsilon + b_{\sigma(k)}), \\
&\quad b_{\sigma(k)} + (K-r)(2\varepsilon + b_{\sigma(k)}),\ a_{\sigma(k+1)}\}, \qquad (3.2)
\end{aligned}$$

where $w_2(k) = \max\{0,\ b_{\sigma(k)} - (2\varepsilon + 4\delta)\}$ is the robot waiting time needed for a component of part $P_{\sigma(k)}$ at machine M_2, and $w_1^r(k+1) = \max\{0,\ a_{\sigma(k+1)} - (\delta +$

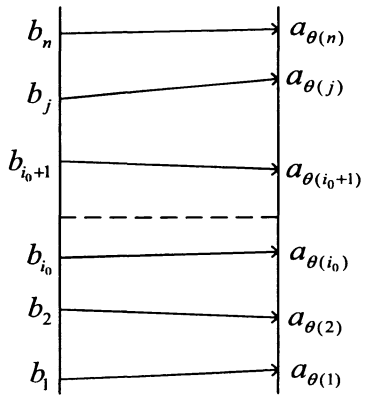

Figure 3.5: The assignment θ

$w_2(k)+\varepsilon+(K-r)(\varepsilon+b_{\sigma(k)}+\varepsilon)+\delta+\varepsilon+2\delta)\} = \max\{0,\ a_{\sigma(k+1)}-(2\varepsilon+4\delta+w_2(k)+(K-r)(2\varepsilon+b_{\sigma(k)}))\}$ is the robot waiting time needed for part $P_{\sigma(k+1)}$ at machine M_1 after the rth component of part $P_{\sigma(k)}$ has been loaded on M_2. The last two equalities above were obtained by substituting these expressions.

Lemma 3.1 *For the K scenarios of robot move cycle type S_2, $T_2^1 \le T_2^2 \le \cdots \le T_2^K$.*

Proof. Consider two scenarios S_2^r and S_2^{r+1} $(r = 1, 2, \ldots, K-1)$. Using (3.2), we have $T_2^{r+1} = 4\varepsilon+4\delta+r(2\varepsilon+b_{\sigma(k)})+\max\{2\varepsilon+4\delta+(K-r-1)(2\varepsilon+b_{\sigma(k)}),\ b_{\sigma(k)}+(K-r-1)(2\varepsilon+b_{\sigma(k)}),\ a_{\sigma(k+1)}\} = 4\varepsilon+4\delta+(r-1)(2\varepsilon+b_{\sigma(k)})+\max\{2\varepsilon+4\delta+(K-r)(2\varepsilon+b_{\sigma(k)}),\ b_{\sigma(k)}+(K-r)(2\varepsilon+b_{\sigma(k)}),\ a_{\sigma(k+1)}+(2\varepsilon+b_{\sigma(k)})\}$.

Since $\max\{2\varepsilon+4\delta+(K-r)(2\varepsilon+b_{\sigma(k)}),\ b_{\sigma(k)}+(K-r)(2\varepsilon+b_{\sigma(k)}),\ a_{\sigma(k+1)}+(2\varepsilon+b_{\sigma(k)})\} \ge \max\{2\varepsilon+4\delta+(K-r)(2\varepsilon+b_{\sigma(k)}),\ b_{\sigma(k)}+(K-r)(2\varepsilon+b_{\sigma(k)}),\ a_{\sigma(k+1)}\}$, we obtain $T_2^{r+1} \ge T_2^r$ $(r = 1, 2, \ldots, K-1)$. This completes our proof. □

The above lemma implies that the best action is to load a new part on machine M_1 for processing as early as possible in an S_2-type robot move cycle, which makes sense intuitively. Thus to find the minimum cycle time for producing an MPS, we only need to consider robot move cycles S_1 and S_2^1.

3.3 Cycle Time Minimization

In this section, we restrict our attention to the class of cyclic schedules. We show first that a single robot move cycle (S_1 or S_2^1) is optimal for cycle time minimization under certain sufficient conditions. Without loss of generality, let us assume that the b_i-s are in non-decreasing order, i.e., $b_i \le b_{i+1}$, and let θ be an assignment (ordering) for which $a_{\theta(i)} \le a_{\theta(i+1)}$. In the rest of this section θ will be used to denote this assignment. A depiction of θ appears in Figure 3.5. Thus $b_i + a_{\theta(i)} \le b_{i+1} + a_{\theta(i+1)}$ for $i = 1, 2, \ldots, n-1$.

Theorem 3.2 *The cycle time for producing an MPS is minimized under cycle* S_1 *if* $b_n + a_{\theta(n)} \leq 2\delta$, *and under cycle* S_2^1 *if* $b_1 + a_{\theta(1)} \geq 2\delta$.

Proof. Suppose $\sigma(k) = i$ and $\sigma(k+1) = j$. From Equations (3.1) and (3.2), we know that $T_1 = 6\varepsilon + 6\delta + b_i + (K-1)(2\varepsilon + b_i) + a_j$ and $T_2^1 = 4\varepsilon + 4\delta + \max\{2\varepsilon + 4\delta + (K-1)(2\varepsilon + b_i),\ b_i + (K-1)(2\varepsilon + b_i),\ a_j\}$.

- If $b_n + a_{\theta(n)} \leq 2\delta \Rightarrow T_1 = 6\varepsilon + 6\delta + b_i + (K-1)(2\varepsilon + b_i) + a_j \leq 6\varepsilon + 8\delta + (K-1)(2\varepsilon + b_i) \leq T_2^1 \Rightarrow$ cycle S_1 is optimal;
- If $b_1 + a_{\theta(1)} \geq 2\delta \Rightarrow T_1 = 6\varepsilon + 6\delta + b_i + (K-1)(2\varepsilon + b_i) + a_j \geq 6\varepsilon + 8\delta + (K-1)(2\varepsilon + b_i) \Rightarrow T_1 \geq T_2^1 \Rightarrow$ cycle S_2^1 is optimal. □

Remark As $a_1 = a_2 = \ldots = a_n = a$ and $b_1 = b_2 = \ldots = b_n = b$ for identical parts, one of the two sufficient conditions is always satisfied in this case. Thus, we can see from the above theorem that the repetition of the best one-unit cycle dominates all other policies for *identical* parts in this robotic cell.

Generally, (i.e., when sufficient conditions do not apply) following a single robot move cycle for different parts is sub-optimal for the cycle time minimization problem, i.e., the cycle time can be improved by a mix of robot move cycles S_1 and S_2^1. We state this in the following lemma.

Lemma 3.3 *Following a single robot move cycle* (S_1 *or* S_2^1) *is not necessarily optimal for* $RC2|1\text{-}K|C_t$.

Proof. When $K = 1$, our problem is equivalent to the cycle time minimization problem with regular processing in the two-machine robotic cell. Hall, Kamoun, and Sriskandarajah (1997) have proved that a single robot move cycle is not optimal for a specific example. □

As a consequence, we must consider a mix of robot move cycles to find the minimum cycle time. Recall that there are only two robot move cycles (S_1 and S_2^1) which need to be considered. For these two cycles, the only state in which switching from S_1 to S_2^1 or *vice versa* is possible, without wasteful robot moves, is the state $\mathcal{G} = (\varnothing, \Omega, M_2^{(1)-})$.

Using Equations (3.1) and (3.2), the minimum possible elapsed time between two consecutive states $\mathcal{G}$ involving the processing of parts $P_{i=\sigma(k)}$ and $P_{j=\sigma(k+1)}$ will be $T_{ij} = \min\{T_1, T_2^1\}$, i.e.,

$$\begin{aligned} T_{ij} = &\ \min\{6\varepsilon + 6\delta + b_i + (K-1)(2\varepsilon + b_i) + a_j, \\ &\ 4\varepsilon + 4\delta + \max\{2\varepsilon + 4\delta + (K-1)(2\varepsilon + b_i),\ b_i + (K-1)(2\varepsilon + b_i),\ a_j\}\} \\ = &\ 4\varepsilon + 4\delta + \min\{2\varepsilon + 2\delta + b_i + (K-1)(2\varepsilon + b_i) + a_j, \\ &\ \max\{(K-1)(2\varepsilon + b_i) + \max\{2\varepsilon + 4\delta, b_i\},\ a_j\}\}. \end{aligned}$$

Since $4\varepsilon+4\delta$ is constant, T_{ij} can be viewed as this constant plus the distance between city i and j in a TSP with the distance matrix $C' = (c'_{ij}) = \min\{2\varepsilon + 2\delta + b_i + (K-1)(2\varepsilon + b_i) + a_j,\ \max\{(K-1)(2\varepsilon + b_i) + \max\{2\varepsilon + 4\delta,\ b_i\},\ a_j\}\}$. As a result, the scheduling problem is equivalent to the problem of finding an optimal tour for this n-city TSP.

Theorem 3.4 *The problem of finding the optimal robot move cycle and part processing sequence to minimize the cycle time in RC2\|1-K\|C_t can be formulated as a TSP.*

Now consider the matrix $C = (c_{ij}) = \min\{\lambda + \eta + Kb_i + a_j,\ \max\{\lambda + (K-1)b_i + \max\{\eta + \rho, b_i\},\ a_j\}\}$, where λ, η, ρ and K are given non-negative constants. It is easy to see that when $\lambda = 2(K-1)\varepsilon$, $\eta = 2\varepsilon + 2\delta$, and $\rho = 2\delta$, this matrix is the same as the distance matrix C' defined above. Next we will establish that the TSP with a distance matrix of the form of C is polynomially solvable.

Define $B_i = \lambda+(K-1)b_i+\max\{\eta+\rho, b_i\}$ and $F_i = \lambda+\eta+Kb_i$ for $i = 1, 2, \ldots, n$. It is easy to verify that the distance matrix $C^\theta = (c_{i\theta(j)}) = \min\{\lambda + \eta + Kb_i + a_{\theta(j)},\ \max\{\lambda + (K-1)b_i + \max\{\eta + \rho, b_i\},\ a_{\theta(j)}\}\}$ can be calculated as a sub-matrix

$$c_{i\theta(j)} = F_i + a_{\theta(j)} \tag{3.3}$$

of sum type if $b_i + a_{\theta(j)} \le \rho$ for any $1 \le i, j \le n$, and a sub-matrix

$$c_{i\theta(j)} = \max\{B_i, a_{\theta(j)}\} \tag{3.4}$$

of Gilmore–Gomory type if $b_i + a_{\theta(j)} > \rho$ for any $1 \le i, j \le n$. Note that $b_i + a_{\theta(i)} \le b_{i+1} + a_{\theta(i+1)}$ for all $i = 1, 2, \ldots, n-1$, and consider the following cases:

- If $b_n + a_{\theta(n)} \le \rho$, then $C^\theta = (c_{i\theta(j)}) = F_i + a_{\theta(j)}$. In this case, every tour has the same length $\sum_{i=1}^n F_i + \sum_{j=1}^n a_{\theta(j)} = n(\lambda + \eta) + K\sum_{i=1}^n b_i + \sum_{j=1}^n a_j$, and thus any tour is optimal.
- If $b_1 + a_{\theta(1)} > \rho$, then $C^\theta = (c_{i\theta(j)}) = \max\{B_i, a_{\theta(j)}\}$, in which case the TSP can be solved by the Gilmore–Gomory algorithm.

Therefore, the only remaining case of interest is when there exists an i_0, $1 \le i_0 \le n-1$, such that $b_{i_0} + a_{\theta(i_0)} \le \rho < b_{i_0+1} + a_{\theta(i_0+1)}$. The remainder of the section deals with this case. Note that the assignment θ in this case, divided by a base line (see the dashed line of Figure 3.5), is composed of two parts: the part including arcs $(i, \theta(i))$ for $i \le i_0$ and the part including arcs $(i, \theta(i))$ for $i \ge i_0 + 1$.

An $n \times n$ matrix $C = (c_{ij})$ is a *Monge (distribution) matrix*, if it fulfills the so-called Monge property:

$$c_{ij} + c_{i'j'} \le c_{ij'} + c_{i'j} \text{ for all } 1 \le i < i' \le n \text{ and } 1 \le j < j' \le n. \tag{3.5}$$

Furthermore, $C = (c_{ij})$ is a *permuted Monge (distribution) matrix*, if there exists a permutation ϕ such that $C^{\phi} = (c_{i\phi(j)})$ is a Monge matrix. The Monge property has received considerable attention in combinatorial optimization. It often makes otherwise computationally hard problems solvable in polynomial time. For a thorough survey of the extensive results on this subject, the interested reader is referred to Burkard, Klinz, and Rudolf (1996). The following crucial lemma shows that our matrix belongs to this class.

Lemma 3.5 *The assignment θ is a permutation for which $C^{\theta} = (c_{i\theta(j)})$ is a Monge matrix. In other words, $C = (c_{ij}) = \min\{\lambda + \eta + Kb_i + a_j,\ \max\{\lambda + (K-1)b_i + \max\{\eta + \rho, b_i\},\ a_j\}\}$ is a permuted Monge matrix.*

Proof. To show that $C^{\theta} = (c_{i\theta(j)}) = (d_{ij})$ is a Monge matrix, we have to prove that $\Delta_{ii'jj'} = d_{ij} + d_{i'j'} - d_{ij'} - d_{i'j} \leq 0$ for all $1 \leq i < i' \leq n$ and $1 \leq j < j' \leq n$.

Recall that the b_i-s are in non-decreasing order and θ is the assignment for which $a_{\theta(j)} \leq a_{\theta(j+1)}$. If d_{kl} has been calculated according to Equation (3.4), then all d_{pl} with $p > k$ and all d_{kq} with $q > l$ have to be calculated with (3.4) as well. This clearly implies that $\Delta_{kplq} \leq 0$ for these cases, because max is a Monge function. In other words, an $n \times n$ matrix $C = (c_{ij})$ with $c_{ij} = \max\{e_i, f_j\}$ satisfies the Monge property, i.e., inequality (3.5). Similarly, if $d_{k'l'}$ has been calculated according to Equation (3.3), then all $d_{pl'}$ with $p < k'$ and all $d_{k'q}$ with $q < l'$ have to be calculated with (3.3) as well, which implies $\Delta_{pk'ql'} = 0$. Therefore, it suffices to consider $\Delta_{ii'jj'}$ when d_{ij} is calculated by Equation (3.3) and $d_{i'j'}$ is calculated by Equation (3.4). Four possible scenarios for $d_{ij'}$ and $d_{i'j}$ are thus left to be considered.

Case I: Both $d_{ij'}$ and $d_{i'j}$ are calculated by (3.3).

In this case, $\Delta_{ii'jj'} = F_i + a_{\theta(j)} + \max\{B_{i'}, a_{\theta(j')}\} - F_i - a_{\theta(j')} - F_{i'} - a_{\theta(j)}$. Notice that (d_{ij}) is initially defined as $\min\{(3.3), (3.4)\}$. If we substitute for the term $\max\{B_{i'}, a_{\theta(j')}\}$ in $\Delta_{ii'jj'}$ the larger value $F_{i'} + a_{\theta(j')}$, we obtain 0 as a result. This yields $\Delta_{ii'jj'} \leq 0$.

Case II: Both $d_{ij'}$ and $d_{i'j}$ are calculated by (3.4).

In this case, $\Delta_{ii'jj'} = F_i + a_{\theta(j)} + \max\{B_{i'}, a_{\theta(j')}\} - \max\{B_i, a_{\theta(j')}\} - \max\{B_{i'}, a_{\theta(j)}\}$. If we replace $F_i + a_{\theta(j)}$ by the larger value $\max\{B_i, a_{\theta(j)}\}$, we still have $\Delta_{ii'jj'}$ non-positive because max is a Monge function. Hence, we get $\Delta_{ii'jj'} \leq 0$ as well.

Case III: $d_{ij'}$ is calculated by (3.3) and $d_{i'j}$ is calculated by (3.4).

In this case, $\Delta_{ii'jj'} = F_i + a_{\theta(j)} + \max\{B_{i'}, a_{\theta(j')}\} - F_i - a_{\theta(j')} - \max\{B_{i'}, a_{\theta(j)}\} = \max\{a_{\theta(j)} + B_{i'}, a_{\theta(j)} + a_{\theta(j')}\} - \max\{a_{\theta(j')} + B_{i'}, a_{\theta(j')} + a_{\theta(j)}\}$. As $a_{\theta(j)} + B_{i'} \leq a_{\theta(j')} + B_{i'}$, we have $\Delta_{ii'jj'} \leq 0$.

Case IV: $d_{ij'}$ is calculated by (3.4) and $d_{i'j}$ is calculated by (3.3).

In this case, $\Delta_{ii'jj'} = F_i + a_{\theta(j)} + \max\{B_{i'}, a_{\theta(j')}\} - \max\{B_i, a_{\theta(j')}\} - F_{i'} - a_{\theta(j)} = \max\{F_i + B_{i'}, F_i + a_{\theta(j')}\} - \max\{F_{i'} + B_i, F_{i'} + a_{\theta(j')}\}$.

By definition,

$$\begin{aligned} F_i + B_{i'} &= \lambda + \eta + Kb_i + \lambda + (K-1)b_{i'} + \max\{\eta + \rho, b_{i'}\} \\ &= \lambda + \eta + Kb_{i'} + \lambda + (K-1)b_i + \max\{\eta + \rho, b_{i'}\} + b_i - b_{i'} \\ &= \lambda + \eta + Kb_{i'} + \lambda + (K-1)b_i + \max\{\eta + \rho + b_i - b_{i'}, b_i\}. \end{aligned}$$

As $b_i \leq b_{i'}$, we obtain $F_i + B_{i'} \leq \lambda + \eta + Kb_{i'} + \lambda + (K-1)b_i + \max\{\eta + \rho, b_i\} = F_{i'} + B_i$. Moreover, it is clear that $F_i + a_{\theta(j')} \leq F_{i'} + a_{\theta(j')}$. These imply $\Delta_{ii'jj'} \leq 0$ again.

This completes our proof. □

The distance matrix being permuted Monge does not automatically lead to an efficient solution for a TSP, since it is known that it remains $\mathcal{NP}$-hard even on this restricted class of matrices (Sarvanov 1980). On the other hand, in Chapter 5 we will investigate special cases of the TSP on so-called *b-decomposable* (permuted Monge) *matrices*. Since our matrix $D = C^\theta$ can be partitioned by an index i_0 ($1 \leq i_0 \leq n-1$) into two sub-matrices $D' = (d'_{ij}) = F_i + a_{\theta(j)}$ for $i, j \leq i_0$ and $D'' = (d''_{ij}) = \max\{B_i, a_{\theta(j)}\}$ for $i, j > i_0$, such that D' is a sum matrix and D'' is a Gilmore–Gomory matrix, it belongs to the class of b-decomposable matrices (cf. Section 5.3.1). Assuming that the permutation for a b-decomposable matrix is given a priori, we will establish that the TSP is solvable in $O(n \log n)$ time on such matrices. For details of the algorithm and more general cases, we refer the reader to Chapter 5. As for our case the permutation (assignment) θ can be found by sorting the b_i and a_i values in such a way that $b_i \leq b_{i+1}$ and $a_{\theta(i)} \leq a_{\theta(i+1)}$, which requires $O(n \log n)$ time, it follows that our TSP is solvable in $O(n \log n)$ time. For the scheduling problem, this leads to the following theorem.

Theorem 3.6 *The RC2|1-K|C_t problem can be solved optimally in $O(n \log n)$ time.*

3.4 Makespan Minimization

In this section we consider the makespan version of the robotic-cell problem, which is denoted by *RC*2|1-*K*|$C_{\max}$. In general, the makespan and cycle time are two different performance measures. Beginning from time zero, the *makespan* of a schedule is the time at which the last finished part is dropped at the output station. Whereas, the cycle time of a schedule is the elapsed time between completions of successive MPSs. Consequently, a part processing sequence that is optimal for a problem with the cycle time objective is not necessarily optimal for a problem with the makespan objective, and vice versa. (The same example as in the proof of Lemma 2.6 in Section 2.3.2 can be used to demonstrate this even for $K = 1$.)

As following a single robot move cycle is not optimal generally, which is shown to be true even for the classical robotic-cell problems with $K = 1$ (cf. Lemma 2.5 in

Section 2.3.2), we consider both robot move cycles S_1 and S_2^1 to find the minimum makespan.

To find the minimum makespan, we know from the discussion preceding Theorem 3.4 that the elapsed time between two consecutive states $\mathcal{G}$ involving the processing of parts $P_{i=\sigma(k)}$ and $P_{j=\sigma(k+1)}$, $k = 1, 2, \ldots, n-1$ is equal to $T_{ij} = \min\{T_1, T_2^1\} = 4\varepsilon + 4\delta + c'_{ij}$, where $c'_{ij} = \min\{2\varepsilon + 2\delta + b_i + (K-1)(2\varepsilon + b_i) + a_j, \max\{(K-1)(2\varepsilon + b_i) + \max\{2\varepsilon + 4\delta, b_i\}, a_j\}\}$. By adding the time required to reach the first occurrence of state $\mathcal{G}$ by part $P_{\sigma(1)}$ and the time needed after the last occurrence of state $\mathcal{G}$ to get part $P_{\sigma(n)}$ finished and delivered to Out, the makespan under part processing sequence $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$ can be calculated as follows. (Here we assume that the robot is available at the input station at time zero.)

$$\begin{aligned} C_{\max}(\sigma) &= \varepsilon + \delta + \varepsilon + a_{\sigma(1)} + \varepsilon + \delta + \varepsilon + \sum_{k=1}^{n-1} T_{\sigma(k)\sigma(k+1)} + b_{\sigma(n)} + \varepsilon \\ &\quad + (K-1)(\varepsilon + b_{\sigma(n)} + \varepsilon) + \delta + \varepsilon \\ &= 6\varepsilon + 3\delta + a_{\sigma(1)} + b_{\sigma(n)} + (K-1)(2\varepsilon + b_{\sigma(n)}) + \sum_{k=1}^{n-1} T_{\sigma(k)\sigma(k+1)} \\ &= (4n+2)\varepsilon + (4n-1)\delta + a_{\sigma(1)} + H(\sigma), \end{aligned}$$

where $H(\sigma) = b_{\sigma(n)} + (K-1)(2\varepsilon + b_{\sigma(n)}) + \sum_{k=1}^{n-1} c'_{\sigma(k)\sigma(k+1)}$.

Since $(4n+2)\varepsilon + (4n-1)\delta$ is constant, it is clear that minimizing the makespan $C_{\max}(\sigma)$ is equivalent to the minimization of $a_{\sigma(1)} + H(\sigma)$. Furthermore, this is equivalent to minimizing $H(\sigma)$ *if* $\sigma(1)$—the first part to be processed—is fixed, i.e., let $\sigma(1) = u$, where $1 \le u \le n$.

Now let us consider an n-city TSP with distance matrix $\overline{C} = (\bar{c}_{ij}) = \min\{2\varepsilon + 2\delta + b_i + (K-1)(2\varepsilon + b_i) + \bar{a}_j, \max\{(K-1)(2\varepsilon + b_i) + \max\{2\varepsilon + 4\delta, b_i\}, \bar{a}_j\}\}$, where $\bar{a}_j = a_j \ge 0$ for $j \in \{1, \ldots, n\} - u$ and $\bar{a}_u = -2\varepsilon - 2\delta \le 0$. Since $\bar{c}_{\sigma(n)u} = b_{\sigma(n)} + (K-1)(2\varepsilon + b_{\sigma(n)})$, the length of the tour $(\sigma(1) = u, \sigma(2), \ldots, \sigma(n))$ is equal to $\sum_{k=1}^{n-1} \bar{c}_{\sigma(k)\sigma(k+1)} + \bar{c}_{\sigma(n)u} = \sum_{k=1}^{n-1} c'_{\sigma(k)\sigma(k+1)} + b_{\sigma(n)} + (K-1)(2\varepsilon + b_{\sigma(n)}) = H(\sigma)$, which is minimized by the optimal tour. Thus the minimum makespan, under the condition that the first part is fixed as $\sigma(1) = u$, can be obtained by solving this TSP. Since the first part $\sigma(1)$ could be $1, 2, \ldots, n$, the minimum makespan can then be found by solving the TSP for every possible choice of the first part, i.e., by letting $u = 1, 2, \ldots, n$. The detailed steps are contained in Algorithm Makespan-1-K.

As the optimal tour length $L^*(u)$, for any given u, can be found in $O(n \log n)$ time by the Algorithm b-Decomposable stated in Section 5.3.1, Algorithm Makespan-1-K takes $O(n^2 \log n)$ time. Thus we have the following result.

Theorem 3.7 *The $RC2|1\text{-}K|C_{\max}$ problem can be solved optimally in $O(n^2 \log n)$ time.*

Algorithm Makespan-1-K
Input: ε, δ, K, a_i and b_i $(i = 1, 2, \ldots, n)$.
1. Let L be a large positive number and $\bar{a}_j = a_j$ $(j = 1, 2, \ldots, n)$.
2. For $u = 1, 2, \ldots, n$ do
   2a. Find the best tour $\sigma^*(u)$ of an n-city TSP with distance matrix $\overline{C} = (\bar{c}_{ij})$ $= \min\{2\varepsilon + 2\delta + b_i + (K-1)(2\varepsilon + b_i) + \bar{a}_j, \max\{(K-1)(2\varepsilon + b_i) + \max\{2\varepsilon + 4\delta, b_i\}, \bar{a}_j\}\}$ and $\bar{a}_u = -2\varepsilon - 2\delta$. Let $L^*(u)$ be the tour length.
   2b. If $L^*(u) + a_u < L$ then $L = L^*(u) + a_u$ and $\sigma^* = \sigma^*(u)$.
3. The minimum makespan $C^*_{\max} = L + (4n+2)\varepsilon + (4n-1)\delta$.
Output: $C^*_{\max}$ and the optimal part processing sequence σ^*.

3.5 Summary

We have studied the problem of scheduling multi-component parts in a two-machine robotic cell. The objective was to find the optimal part processing sequence and robot move cycle which jointly minimize the cycle time or the makespan. We showed that these problems can be formulated as special cases of the TSP. By showing that these special cases of the TSP are solvable in polynomial time, we have presented an $O(n \log n)$ algorithm for the problem with the cycle time objective and an $O(n^2 \log n)$ algorithm for the problem with the makespan objective.

Chapter 4
Cyclic Scheduling in a Job Shop

4.1 Introduction

Hall, Lee, and Posner (2002) studied a two-machine cyclic job shop scheduling problem, where the maximum number of operations for any job is bounded by a constant $k \geq 1$. A typical cyclic schedule consists of the repeated processing of a minimal job (part) set (cf. Section 1.2). Formally, the problem may be formulated as follows.

We are given a minimal job set $\mathcal{J} = \{1, 2, \ldots, n\}$ in a job shop consisting of two machines, M_1 and M_2. Each job $j \in \mathcal{J}$ has m_j operations O_{ij} $(i = 1, 2, \ldots, m_j)$ to be processed in the order $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{m_j j}$ where $m_j \leq k$. Operation O_{ij} has to be performed on a specified machine M_1 or M_2 without interruption for p_{ij} time units, where all p_{ij}'s are assumed to be positive integers. We assume that any two consecutive operations of the same job are to be processed on different machines; for otherwise, they can be combined into a single operation. The machines have unlimited buffer and a job can be stored in the buffer for an unlimited amount of time. Moreover, it is assumed that each machine can handle at most one operation at a time, and each operation can be processed only by one machine. Let $s_i(\sigma)$ and $c_i(\sigma)$ be the earliest start time and the latest completion time, respectively, of any job on machine M_i $(i = 1, 2)$ in a schedule σ for $\mathcal{J}$. We call $r_i(\sigma) = c_i(\sigma) - s_i(\sigma)$ the *running time of* M_i in σ. The *running time of the schedule* σ is defined by $\max_{i=1,2} r_i(\sigma)$. Lee and Posner (1997) establish that in order to minimize the cycle time of a cyclic schedule, we only need to determine an operation sequence for a minimal job set that minimizes the running time of the corresponding schedule for this job set. They also show that the *cycle time of the corresponding cyclic schedule* can be defined as $C_t(\sigma) = \max_{i=1,2} r_i(\sigma)$. The goal of the problem is to find a feasible schedule σ^* that has the minimum cycle time, i.e., $C_t(\sigma^*) = \min_\sigma C_t(\sigma)$. Following the standard notation for scheduling problems, we denote this problem by $J2|m_j \leq k|C_t$.

Hall, Lee, and Posner (2002) described an $O(n)$ algorithm for this problem for the $k = 2$ case. In addition, they showed that the problem becomes $\mathcal{NP}$-hard when $k = 3$ and even strongly $\mathcal{NP}$-hard when $k = 5$. For a special case of the problem $J2|m_j \leq 3|C_t$ which satisfies the condition that there is a "consistent" job order for generating partial schedules, they presented an $O(n^{13} p_{\max}^{11})$ algorithm to

solve it. However, whether the general $J2|m_j \leq k|C_t$ problem can be solved in pseudo-polynomial time when $k = 3$ or 4 remained an open question.

In this chapter, we reexamine the $k = 3$ case, i.e., when each job has at most three operations. We show that $J2|m_j \leq 3|C_t$ can be polynomially reduced to minimizing the makespan in a two-machine reentrant flow shop. Based on this insight, we apply previously known and newly obtained results for the reentrant flow shop problem to $J2|m_j \leq 3|C_t$. We extend previous results for the reentrant flow shop problem by presenting a new pseudo-polynomial algorithm, as well as a fully polynomial time approximation scheme (FPTAS) for certain special cases. This leads to new pseudo-polynomial time solutions for additional special cases of the $J2|m_j \leq 3|C_t$ problem. We also describe a polynomial time approximation algorithm for the general case of $J2|m_j \leq 3|C_t$.

The remainder of the chapter is organized as follows. In Section 4.2, we show that the $J2|m_j \leq 3|C_t$ problem is reducible to the two-machine reentrant flow shop problem, and describe an $O(n \log n)$-time 4/3-approximation algorithm for it. In Section 4.3, we consider special cases of the reentrant flow shop problem and the $J2|m_j \leq 3|C_t$ problem. We present new optimal, and approximate solutions to both problems. A few well-solvable special cases are also distinguished. Finally, Section 4.4 contains our conclusions.

4.2 Problem Analysis

Notice that a job may have one, two, or three operations, and we call a job a *one-operation job*, *two-operation job*, or *three-operation job* accordingly. Based on job processing routes, we divide the job set $\mathcal{J}$ into six subsets: $\mathcal{J}_i$ $(i = 1, 2)$ consisting of the jobs that have to be processed on machine M_i only; $\mathcal{J}_{ij}$ $(i, j = 1, 2$ and $i \neq j)$ consisting of the jobs that have to be first processed on machine M_i and then on M_j; $\mathcal{J}_{121}$ and $\mathcal{J}_{212}$ consisting of the jobs with processing routes $M_1 \rightarrow M_2 \rightarrow M_1$ and $M_2 \rightarrow M_1 \rightarrow M_2$, respectively. Without loss of generality, we assume that the machines have been indexed such that $\sum_{j \in \mathcal{J}_{212}} p_{2j} \geq \sum_{j \in \mathcal{J}_{121}} p_{2j}$.

To attain the optimal schedule for the $J2|m_j \leq 3|C_t$ problem, similarly to Hall, Lee, and Posner (2002), our strategy is first to build a best partial schedule σ'' for the three-operation jobs, i.e., the minimum-cycle-time schedule for $\mathcal{J}_{121} \cup \mathcal{J}_{212}$. Following this, we insert the one- and two-operation jobs into σ'' so that they do not create any additional idle period between operations, thus obtaining a complete schedule for $\mathcal{J}$. As we will demonstrate later, through a careful arrangement, the one- and two-operation jobs can be scheduled separately after the three-operation jobs have been optimally sequenced. Recall that the running time of a machine is the time from when a machine starts any processing of any job until it ends all processing. During its running time, a machine may be idle for some time. We call any such idle period *suspend time* for that machine. The total suspend time of a machine is

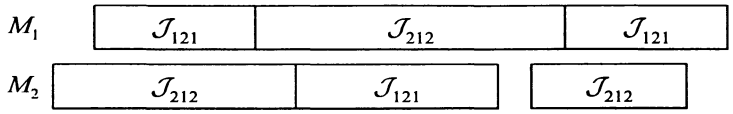

Figure 4.1: An example of the block schedule for $\mathcal{J}_{121} \cup \mathcal{J}_{212}$

different from the total idle time, since the latter also includes times a machine may be idle for before it does any processing. It is easy to see that minimizing the cycle time can be achieved by minimizing the total suspend time of every machine. Our partial schedule σ'' will also have this property. Furthermore, in the insertion of one- and two-operation jobs into the partial schedule σ'', we will schedule these jobs in a way which will reduce the suspend times of both machines M_1 and M_2 in σ'' as much as possible.

We present our solution procedure in more detail now. Let us start with the three-operation jobs. For these jobs, Hall, Lee, and Posner (2002) have identified several important properties, which may be stated in the following theorem. Note that Characteristic b uses the nonrestrictive assumption that $\sum_{j\in\mathcal{J}_{212}} p_{2j} \geq \sum_{j\in\mathcal{J}_{121}} p_{2j}$.

Theorem 4.1 (Hall, Lee, and Posner 2002) *For the problem $J2|m_j = 3|C_t$, there exists an optimal schedule with the following characteristics:*

a. On each machine M_1 and M_2, all the first operations of jobs precede all the second operations of jobs, which precede all the third operations of jobs.

b. All of the second operations on machine M_2 are processed concurrently with the second operations on machine M_1.

c. There is no idle period between any operations on machine M_1. Also, there is no idle period between any pair of first operations, second operations, or third operations on machine M_2.

d. The first, second, and third operations of all jobs in $\mathcal{J}_{121}$ or $\mathcal{J}_{212}$ are processed in the same sequence on each machine.

A *block* of operations is a set of operations that are performed consecutively on a machine without idle period between them. From the above theorem, we know that there exists an optimal schedule for the jobs in $\mathcal{J}_{121} \cup \mathcal{J}_{212}$ where on each machine, the first operations, the second operations, and the third operations each form a block. Furthermore, in this schedule, there is no suspend time on machine M_1, and $\mathcal{J}_{121}$ starts its second operation on M_2 no earlier than the time at which $\mathcal{J}_{212}$ starts its second operation on M_1. Note that when the elapsed time between the completion of the first operations and the start of the third operations of $\mathcal{J}_{212}$ on machine M_2 in a schedule is greater than $\sum_{j\in\mathcal{J}_{121}} p_{2j}$, there is some slack in this schedule so that the second operations of $\mathcal{J}_{121}$ can start earlier or later without affecting the cycle time. Thus, in order to specify a unique starting time for $\mathcal{J}_{121}$, we assume that the set of second operations of $\mathcal{J}_{121}$ starts its processing immediately after the set of first

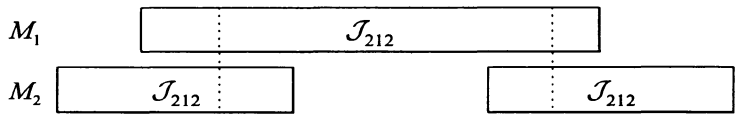

Figure 4.2: The structure of the block schedule σ' for $\mathcal{J}_{212}$

operations of $\mathcal{J}_{212}$ finishes its processing on M_2. Figure 4.1 illustrates an example of such block schedules. Note also that the schedule shown has a positive suspend time on M_2 but no such time on M_1.

Since we know from Theorem 4.1 (Characteristic b) that all of the second operations of the jobs in $\mathcal{J}_{121}$ can be processed concurrently with the second operations of the jobs in $\mathcal{J}_{212}$, it is clear that the timings of the three operations of $\mathcal{J}_{121}$ are independent from each other in such block schedules. As a result, any sequence for the three operations of the jobs in $\mathcal{J}_{121}$ is optimal (Hall, Lee, and Posner 2002).

Let us consider now the sequence for the jobs in $\mathcal{J}_{212}$. Lev and Adiri (1984), Wang, Sethi, and van de Velde (1997), as well as Drobouchevitch and Strusevich (1999) study a two-machine reentrant flow shop problem, where each job j, $j \in \{1, 2, \ldots, n\}$, has a sequence of three operations in the order $O_{1j} \rightarrow O_{2j} \rightarrow O_{3j}$. For each job j, operations O_{1j}, O_{2j}, and O_{3j} have to be processed without preemption along the route $M_1 \rightarrow M_2 \rightarrow M_1$ (or $M_2 \rightarrow M_1 \rightarrow M_2$) for p_{1j}, p_{2j} and p_{3j} time units, respectively. Each machine can only process one operation at a time, and there is unlimited input and output buffer space available for each machine. The objective is to determine a feasible schedule minimizing the makespan or the maximum completion time. We denote this problem by $RF2|\ell = 3|C_{\max}$, where ℓ is the number of operations of a job. Since the maximum running time and the maximum completion time are always reached on the machine processing the first and third operations of the jobs in the two-machine reentrant flow shop, the cycle time objective is equivalent to the makespan objective. Furthermore, the schedule minimizing the makespan will also minimize the total suspend time of each machine. Observe that each job in $\mathcal{J}_{212}$ has the processing route $M_2 \rightarrow M_1 \rightarrow M_2$. It follows that the sequence minimizing the total suspend time on each machine for the jobs in $\mathcal{J}_{212}$ and its corresponding schedule can also be found by solving the $RF2|\ell = 3|C_{\max}$ problem. Let σ' be the best schedule generated for the jobs in $\mathcal{J}_{212}$. The schedule σ' can decompose into blocks too (Wang, Sethi, and van de Velde 1997). More precisely, σ' is composed of three blocks, with each block representing an operation of the jobs in $\mathcal{J}_{212}$. Moreover, machine M_1 has no suspend time in σ', and if there exists any suspend time on machine M_2, it is always between the end of the first operations and the start of the third operations. The structure of the block schedule σ' is depicted in Figure 4.2. (The dashed lines indicate the critical operation set, as defined in Hall, Lee, and Posner (2002).) More properties of the schedule σ' will be discussed in Section 4.3.1.

Once we have the schedule σ', we can insert the three operations of $\mathcal{J}_{121}$ in blocks, and in any order, into σ' to obtain σ''. We simply follow the rules in Theorem

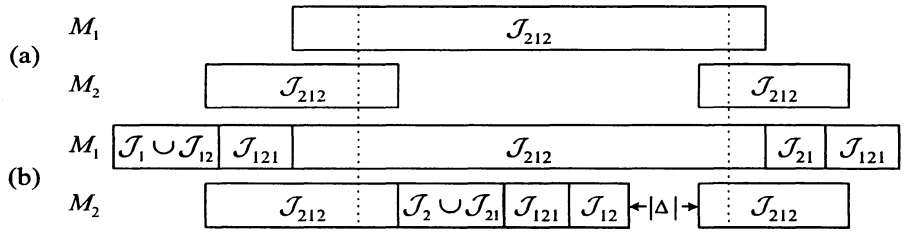

Figure 4.3: (a) The partial schedule σ' for $\mathcal{J}_{212}$; (b) The overall schedule σ^*

4.1. Note that if the suspend time of M_2 in σ' is not long enough, inserting the second operations of $\mathcal{J}_{121}$ into this slot may force us to shift some of the other blocks to the left or the right on M_2. This will be discussed in detail later on, but we describe now how to schedule the one- and two-operation jobs. After the best partial schedule σ'' for the three-operation jobs has been constructed (Figure 4.1 shows the possible structure for σ''), we insert the one- and two-operation jobs into σ'' in blocks so that they never increase the suspend times of M_1 and M_2. Specifically, we

1. insert the first operations of $\mathcal{J}_1 \cup \mathcal{J}_{12}$ in arbitrary order before (or after) the first operations of $\mathcal{J}_{121}$ on machine M_1, insert the second operations of $\mathcal{J}_{12}$ in arbitrary order after the second operations of $\mathcal{J}_{121}$ on machine M_2; and
2. insert the first operations of $\mathcal{J}_2 \cup \mathcal{J}_{21}$ in arbitrary order before the second operations of $\mathcal{J}_{121}$ on machine M_2, insert the second operations of $\mathcal{J}_{21}$ in arbitrary order before (or after) the third operations of $\mathcal{J}_{121}$ on machine M_1.

Let us call the schedule resulting after all these insertions σ^*. Figures 4.3, 4.4, and 4.5 show the different situations which may occur during the construction of σ^*. Let $C_{\max}(\sigma')$ be the length of the best partial schedule σ' for $\mathcal{J}_{212}$, i.e., the schedule generated by solving the $RF2|\ell = 3|C_{\max}$ problem. Define $\Delta = \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_2\cup\mathcal{J}_{21}} p_{1j} + \sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} + \sum_{j\in\mathcal{J}_{212}} p_{3j} - C_{\max}(\sigma')$, which is the difference between the total processing time on machine M_2 and the length of σ'. We are now ready to discuss the properties of the schedule σ^* for $\mathcal{J}$ and show that σ^* is actually an optimal schedule for $J2|m_j \leq 3|C_t$.

Recall that in the partial schedule σ'' machine M_1 has no suspend time. The insertion of one- and two-operation jobs into σ'' would not create any suspend time on machine M_1, because M_1 never has to wait for the arrival of a second operation from $\mathcal{J}_{21}$ or a third operation from $\mathcal{J}_{121}$. Thus M_1 has no suspend time in σ^* either. Therefore, to prove that σ^* is optimal, it suffices to prove the minimality of the running time or the suspend time only for machine M_2. There are two cases to be considered:

Case 1: $\Delta < 0$ (see Figure 4.3). In this case, machine M_2 has $|\Delta|$ units of suspend time between the completion of the second operations of $\mathcal{J}_{12}$ and the start of the third operations of $\mathcal{J}_{212}$ in σ^*. The running time of machine M_2 in σ^* is $C_{\max}(\sigma')$.

As $C_{\max}(\sigma')$ is a lower bound for the running time of M_2, the schedule σ^* is indeed optimal, and its cycle time is given by $C_t^1(\sigma^*) = \max\{C_{\max}(\sigma'), \sum_{j\in\mathcal{J}_1\cup\mathcal{J}_{12}\cup\mathcal{J}_{121}} p_{1j} + \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_{21}} p_{2j} + \sum_{j\in\mathcal{J}_{121}} p_{3j}\}$.

Case 2: $\Delta \geq 0$ (see Figures 4.4 and 4.5). In this case, we will show that there is no suspend time on machine M_2 in σ^*, i.e., M_2 never has to wait for a job from $\mathcal{J}_{212}$ to become available for its third operation. Thus schedule σ^* is obviously optimal, and its cycle time is given by $C_t^2(\sigma^*) = \max\{\sum_{j\in\mathcal{J}_1\cup\mathcal{J}_{12}\cup\mathcal{J}_{121}} p_{1j} + \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_{21}} p_{2j} + \sum_{j\in\mathcal{J}_{121}} p_{3j}, \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_2\cup\mathcal{J}_{21}} p_{1j} + \sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} + \sum_{j\in\mathcal{J}_{212}} p_{3j}\}$. Let t_{23} be the time between the start of the second and third operations in σ'.

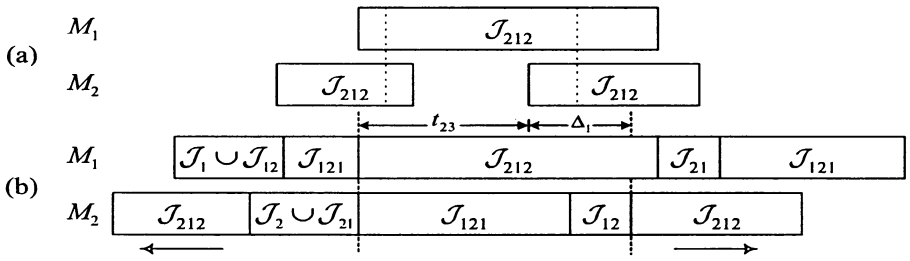

Figure 4.4: (a) The partial schedule σ' for $\mathcal{J}_{212}$; (b) The overall schedule σ^*

If $\sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} > t_{23}$, then σ^* can be constructed as shown in Figure 4.4. In this schedule, the first operations of $\mathcal{J}_{212}$ are shifted to the left so that M_2 has no suspend time after the insertion of all operations, and the second operations of $\mathcal{J}_{121}$ and $\mathcal{J}_{212}$ start simultaneously at the time of the left dashed line in Figure 4.4b. Since $\sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} > t_{23}$, we also have to shift the third operations of $\mathcal{J}_{212}$ to the right so that they will start $\Delta_1 = \sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} - t_{23}$ units of time later than in σ' at the time represented by the right dashed line in Figure 4.4b.

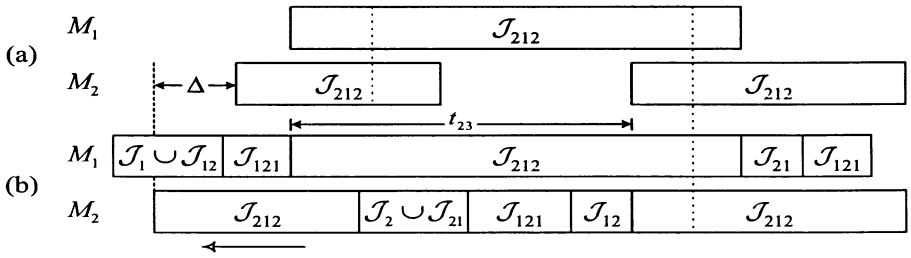

Figure 4.5: (a) The partial schedule σ' for $\mathcal{J}_{212}$; (b) The overall schedule σ^*

If $\sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} \leq t_{23}$, then σ^* can be constructed as shown in Figure 4.5. In this schedule, the first operations of $\mathcal{J}_{212}$ start Δ units of time earlier than in σ'. The shifting of the first operations of $\mathcal{J}_{212}$ to the left by Δ will make exactly the right amount of time available on M_2 to process the operations of $\mathcal{J}_2 \cup \mathcal{J}_{21} \cup \mathcal{J}_{121} \cup \mathcal{J}_{12}$

in a contiguous fashion. Clearly, this schedule is feasible because of the assumption $\sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} \leq t_{23}$.

Finally, combining $C_t^1(\sigma^*)$ and $C_t^2(\sigma^*)$ from the two cases together, the cycle time of the optimal schedule σ^* for $J2|m_j \leq 3|C_t$ can be obtained as

$$C_t(\sigma^*) = \max\left\{ \begin{array}{c} C_{\max}(\sigma'),\ \sum_{j\in\mathcal{J}_1\cup\mathcal{J}_{12}\cup\mathcal{J}_{121}} p_{1j} + \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_{21}} p_{2j} + \sum_{j\in\mathcal{J}_{121}} p_{3j}, \\ \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_2\cup\mathcal{J}_{21}} p_{1j} + \sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} + \sum_{j\in\mathcal{J}_{212}} p_{3j} \end{array} \right\},$$

where $C_{\max}(\sigma')$ is the length of the best partial schedule σ' for $\mathcal{J}_{212}$. Once we have σ', all remaining jobs can be inserted into σ' in $O(n)$ time. Thus we have proved the following result.

Theorem 4.2 *Consider the subset of jobs $\mathcal{J}_{212}$ in a $J2|m_j \leq 3|C_t$ problem. If σ' is an optimal schedule for the corresponding $RF2|\ell = 3|C_{\max}$ problem on the job set $\mathcal{J}_{212}$, then we can construct an optimal schedule σ^* from σ' for $J2|m_j \leq 3|C_t$ in linear time.*

The above theorem tells us that, to find the optimal solution for the $J2|m_j \leq 3|C_t$ problem we need to consider the sequence only for the jobs in $\mathcal{J}_{212}$, which can then be solved as an $RF2|\ell = 3|C_{\max}$ problem. Unfortunately, the problem $RF2|\ell = 3|C_{\max}$ has been shown to be $\mathcal{NP}$-hard by Lev and Adiri (1984), but whether it is $\mathcal{NP}$-hard in the ordinary or strong sense remains an open question.

Consider an $\mathcal{NP}$-hard cost-minimization problem Π. An algorithm $\mathcal{A}$ is said to be a ρ-approximation algorithm for problem Π, if for any instance I of Π it always delivers a feasible solution whose cost is at most ρ times the optimal cost. In this case, the value ρ ($\rho > 1$) is called the performance guarantee or the worst-case performance ratio of the approximation algorithm $\mathcal{A}$.

Drobouchevitch and Strusevich (1999) have presented an $O(n \log n)$ time approximation algorithm with a worst-case performance ratio of 4/3 for $RF2|\ell = 3|C_{\max}$. Thus, we can use this approximation algorithm to find a good approximation of the best partial schedule for $\mathcal{J}_{212}$. Then a 4/3-approximation algorithm for $J2|m_j \leq 3|C_t$ may be simply developed as follows:

1. Apply the Drobouchevitch–Strusevich algorithm to find a good partial schedule $\widehat{\sigma}$ for $\mathcal{J}_{212}$;
2. Follow the steps as shown previously to insert all jobs in $\mathcal{J} - \mathcal{J}_{212}$ into $\widehat{\sigma}$.

Let $\overline{\sigma}$ be the resulting whole schedule generated in Step 2. It is clear that the cycle time of $\overline{\sigma}$ is given by

$$C_t(\overline{\sigma}) = \max\left\{ \begin{array}{c} C_{\max}(\widehat{\sigma}),\ \sum_{j\in\mathcal{J}_1\cup\mathcal{J}_{12}\cup\mathcal{J}_{121}} p_{1j} + \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_{21}} p_{2j} + \sum_{j\in\mathcal{J}_{121}} p_{3j}, \\ \sum_{j\in\mathcal{J}_{212}\cup\mathcal{J}_2\cup\mathcal{J}_{21}} p_{1j} + \sum_{j\in\mathcal{J}_{121}\cup\mathcal{J}_{12}} p_{2j} + \sum_{j\in\mathcal{J}_{212}} p_{3j} \end{array} \right\},$$

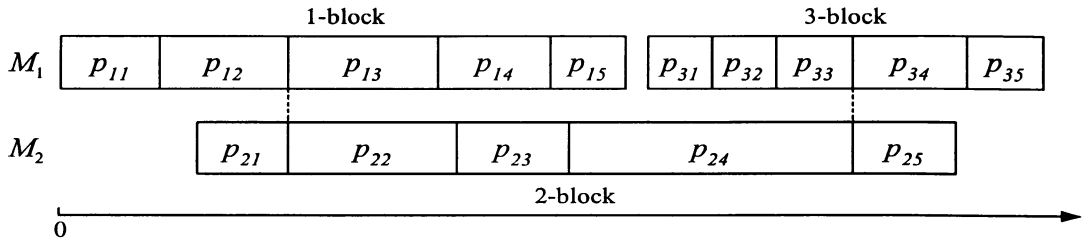

Figure 4.6: A no-passing block schedule for $RF2|\ell = 3|C_{\max}$

where $C_{\max}(\widehat{\sigma})$ is the length of the partial schedule $\widehat{\sigma}$.

As the Drobouchevitch–Strusevich algorithm is a 4/3-approximation algorithm for $RF2|\ell = 3|C_{\max}$, we have $C_{\max}(\widehat{\sigma}) \leq \frac{4}{3}C_{\max}(\sigma')$. Substituting this inequality into $C_t(\overline{\sigma})$ and comparing $C_t(\overline{\sigma})$ with $C_t(\sigma^*)$, it is easy to see that $C_t(\overline{\sigma}) \leq \frac{4}{3}C_t(\sigma^*)$. Since the Drobouchevitch–Strusevich algorithm runs in $O(n \log n)$ time and Step 2 can be performed in $O(n)$ time, we obtain the following result.

Theorem 4.3 *There exists an $O(n \log n)$-time 4/3-approximation algorithm for the problem $J2|m_j \leq 3|C_t$.*

4.3 Special Cases

For the $RF2|\ell = 3|C_{\max}$ problem, Wang, Sethi, and van de Velde (1997) have studied the special case when the first and the last operations of any job are identical, i.e., $p_{1j} = p_{3j}$ for $j = 1, 2, \ldots, n$. The $RF2|\ell = 3|C_{\max}$ problem is still $\mathcal{NP}$-hard for this restricted case (Lev and Adiri 1984; Hall, Lee, and Posner 2002), but they developed a pseudo-polynomial time algorithm to solve it. Their dynamic programming algorithm runs in $O(n^5 p_{\max}^3)$ time and $O(n^3 p_{\max}^2)$ space, where $p_{\max} = \max_{1 \leq i \leq 3, 1 \leq j \leq n} p_{ij}$. In this section, we show that additional cases can be solved in pseudo-polynomial time. We also describe an FPTAS for them.

4.3.1 Pseudo-polynomial algorithm

A schedule is called *no-passing* (or *permutation*) if all jobs go through the machines in the same order. For the problem $RF2|\ell = 3|C_{\max}$, Wang, Sethi, and van de Velde (1997) have shown that it suffices to consider only *no-passing block* (also termed *compact*) schedules which are composed of three blocks, with each block representing an operation of the jobs. Thus, throughout this section our analysis is restricted to such schedules. Figure 4.6 exhibits a no-passing block schedule with the three blocks on five jobs.

Consider any no-passing block schedule ψ that is feasible for the problem $RF2|\ell = 3|C_{\max}$ with job set $\mathcal{N} = \{1, 2, \ldots, n\}$. Suppose ψ is associated with the

permutation (sequence) $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of $\mathcal{N}$. Let a, b, and c denote the total processing time of the 1st, 2nd, and 3rd operation of all the jobs, respectively, i.e., $a = \sum_{j=1}^{n} p_{1j}$, $b = \sum_{j=1}^{n} p_{2j}$, and $c = \sum_{j=1}^{n} p_{3j}$. Then the makespan of the schedule ψ can be derived as

$$C_{\max}(\psi) = \max\{a + c, L(\psi)\},$$

where

$$L(\psi) = \max_{1 \le \mu, \nu \le n} \left\{ \sum_{j=1}^{\mu} p_{1\pi(j)} + \sum_{j=\mu}^{\nu} p_{2\pi(j)} + \sum_{j=\nu}^{n} p_{3\pi(j)} \right\}. \tag{4.1}$$

If $C_{\max}(\psi) = L(\psi)$ and $L(\psi)$ is attained by $\mu = u$ and $\nu = v$, then the jobs $\pi(u)$ and $\pi(v)$ are called *critical* in ψ (Drobouchevitch and Strusevich 1999). For example, jobs 2 and 4 are critical in the schedule shown in Figure 4.6, and the makespan of the schedule is given by $\sum_{j=1}^{2} p_{1j} + \sum_{j=2}^{4} p_{2j} + \sum_{j=4}^{5} p_{3j}$. The schedule ψ is uniquely determined by the permutation π, except when $C_{\max}(\psi) = a + c$ there may be some slack in ψ such that the 2-block (i.e., the block representing the 2nd operation of the jobs) can be shifted to the left or the right without affecting the makespan. Thus, in order to specify a unique starting time for each block, we assume that no block in ψ can start processing earlier than required to realize the makespan of the schedule. (For ease of exposition, in the rest of the chapter, we shall denote a schedule by its job processing sequence when this leads to no ambiguity.)

Let $S_{ij}(\psi)$ and $C_{ij}(\psi)$ be the start time and completion time of operation O_{ij} in the schedule ψ, respectively. (Note that $C_{ij}(\psi) = S_{ij}(\psi) + p_{ij}$.) We observe that there exists a unique job k^* in any schedule ψ, such that

$$k^* = \psi\left(\arg\min_i \{S_{2,\psi(i)}(\psi) + p_{2,\psi(i)} > a\}\right),$$

where $\psi(i)$ denotes the ith job in ψ, $i = 1, 2, \ldots, n$. In other words, k^* is the first job in ψ whose second operation is finished later than a. Using job k^* as a *partition job*, the job set $\mathcal{N}$ can be partitioned into three subsets with respect to ψ: $\mathcal{N}_1 = \{j | j \prec k^*\}$, $\mathcal{N}_2 = \{j | j \succ k^*\}$, and $\{k^*\}$, where $j \prec k^*$ and $j \succ k^*$ mean that j precedes k^* and j follows k^*, respectively. Here, $\mathcal{N}_1$ is referred to as the set of *left*-jobs, i.e., the jobs scheduled before the partition job k^*, and $\mathcal{N}_2$ is referred to as the set of *right*-jobs, i.e., the jobs scheduled after the partition job k^*. For example, in Figure 4.6, job 4 is the partition job with $\mathcal{N}_1 = \{1, 2, 3\}$ and $\mathcal{N}_2 = \{5\}$. Thus, the schedule ψ can be represented by an ordered $\mathcal{N}_1$ followed by job k^* and then by an ordered $\mathcal{N}_2$. Observe that the schedule ψ can also be expressed as the composition of three partial schedules along with the partition $(\mathcal{N}_1, k^*, \mathcal{N}_2)$ of $\mathcal{N}$. More precisely, the first partial schedule, ψ_1, consists of the 1-operations (i.e., 1st operations) and the 2-operations of the jobs in $\mathcal{N}_1$; the second partial schedule, ψ_2, consists of the 2-operations and the 3-operations of the jobs in $\mathcal{N}_2$; and the third partial schedule, ψ_3, consists of

the operations of job k^* and the operations on M_1 scheduled between O_{1k^*} and O_{3k^*}. Since both ψ_1 and ψ_2 involve only two machines, these partial schedules can be viewed as schedules in a classical two-machine flow shop. Recall that a schedule constructed by *Johnson's rule* (Johnson 1954) is optimal for the makespan minimization problem in the classical two-machine flow shop, which is denoted by $F2||C_{\max}$. Let the triple $(\mathcal{Q}, \mathbf{s}, \mathbf{t})$ be an instance of $F2||C_{\max}$ with a job set $\mathcal{Q}$ and processing time vectors $\mathbf{s}$ for the first machine and $\mathbf{t}$ for the second machine. Johnson's rule can be simply described as follows.

Johnson's Rule for $(\mathcal{Q}, \mathbf{s}, \mathbf{t})$:
1. Partition $\mathcal{Q}$ into two subsets U and W with U containing all the jobs with $s_j \leq t_j$ and W all the jobs with $s_j > t_j$.
2. Sort U in non-decreasing order of s_j and W in non-increasing order of t_j.
3. Output the ordered U followed by the ordered W as the optimal job sequence.

Let $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ be the processing time vector of the 1st, 2nd, and 3rd operations of all the jobs in $\mathcal{N}$, respectively. With respect to the jobs in $\mathcal{N}_1$ and $\mathcal{N}_2$, Wang, Sethi, and van de Velde (1997) have established the following property for any instance of $RF2|\ell = 3|C_{\max}$.

Theorem 4.4 (Wang, Sethi, and van de Velde 1997) *There exists an optimal schedule consisting of a partition $(\mathcal{N}_1, k^*, \mathcal{N}_2)$ of $\mathcal{N}$ such that the jobs in $\mathcal{N}_1$ are sequenced according to Johnson's rule for $(\mathcal{N}_1, \mathbf{p}_1, \mathbf{p}_2)$ and the jobs in $\mathcal{N}_2$ are sequenced according to Johnson's rule for $(\mathcal{N}_2, \mathbf{p}_2, \mathbf{p}_3)$, where k^* is the partition job.*

A no-passing block schedule ψ for $RF2|\ell = 3|C_{\max}$ specifies a unique start time for the 2-block. Therefore, we can guess an integer time point s, where $0 \leq s \leq a$, at which the 2-block starts. Then we enumerate all feasible schedules (there might be none) for the jobs such that their 1-operations are scheduled in the time interval $[0, a]$ and their 2-operations are scheduled in the time interval $[s, s+b]$. A feasible schedule ψ must satisfy $S_{2j}(\psi) \geq C_{1j}(\psi)$ for each job $j \in \mathcal{N}$. Since we cannot predetermine the time at which the 3-block starts, we temporarily put all the 3-operations in the time interval $[a+b, a+b+c]$, see Figure 4.7. When a feasible schedule is found, the 3-block will be shifted to the left as much as possible to compute the makespan. To preserve the feasibility, the start time of O_{3j} after the shift must be not earlier than the completion time of O_{2j}. Hence, job j can be shifted to the left by at most $\delta_j(\psi) = S_{3j}(\psi) - C_{2j}(\psi)$ units of time. (Note that any left-job can be shifted to the left by exactly b units of time as its 2-operation is always completed no later than time point a.) This shift is called the *slack of job j* in the schedule ψ. We also define the *slack of a right-job set $\mathcal{T}$* in ψ as $\min_{j\in\mathcal{T}} \delta_j(\psi)$. Thus the 3-block can only be shifted to the left by $\Delta(\psi) = \min_{j\in\mathcal{N}}\{b, \delta_j(\psi)\}$ units of time. (Note that for any left-job $j' \in \mathcal{N}$, $\delta_{j'}(\psi) = S_{3j'}(\psi) - C_{2j'}(\psi) \geq b$.) We call $\Delta(\psi)$ the *slack of the schedule* ψ and the makespan of ψ is then equal to $a+b+c-\Delta(\psi)$. Therefore, for a given

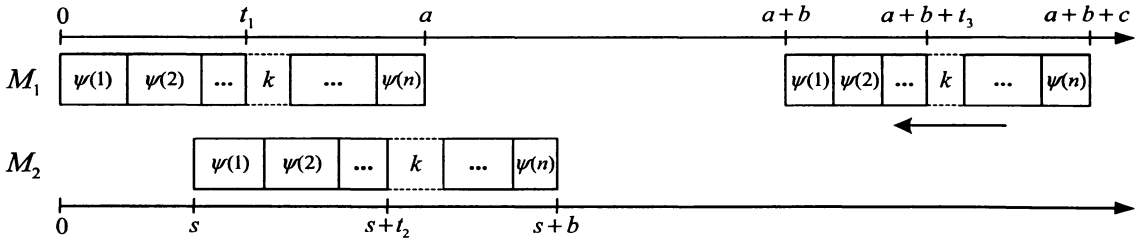

Figure 4.7: A schedule associated with $F_n^{k,s}(t_1, t_2, t_3)$

time point s, the best schedule is the one that maximizes $\Delta(\psi)$. This *insight* serves as the basis of our new solution for special cases of $RF2|\ell = 3|C_{\max}$, as we will show that $\Delta(\psi)$ can be recursively calculated. Naturally, we will generate partial schedules during the enumeration process, and the slack in a partial schedule can be calculated likewise.

From Theorem 4.4, we know that there is an optimal job processing sequence for the problem $RF2|\ell = 3|C_{\max}$ which may be regarded as a concatenation of three subsets of jobs, namely,

1) the left-jobs in Johnson's order for $(\mathbf{p}_1, \mathbf{p}_2)$, followed by
2) a partition job, followed by
3) the right-jobs in Johnson's order for $(\mathbf{p}_2, \mathbf{p}_3)$.

Without loss of generality, let us assume hereinafter that the jobs in $\mathcal{N}$ have been re-indexed according to Johnson's rule for the first two operations, i.e., $(\mathbf{p}_1, \mathbf{p}_2)$. Let $\phi = (\phi(1), \phi(2), \ldots, \phi(n))$ denote the ordering of $\mathcal{N}$ according to Johnson's rule for the last two operations, i.e., $(\mathbf{p}_2, \mathbf{p}_3)$. Next we show that the $RF2|\ell = 3|C_{\max}$ problem is pseudo-polynomially solvable if

$$\phi = (n, n-1, \ldots, 1) \tag{4.2}$$

or

$$\phi = (1, 2, \ldots, n). \tag{4.3}$$

Condition/Equation (4.2) or (4.3) implies that the two orderings of $\mathcal{N}$, derived by applying Johnson's rule for the first and the last two operations, are in the reverse or in the same order, respectively. It is easy to verify that when $p_{1j} = p_{3j}$ for $j = 1, 2, \ldots, n$, we have $\phi = (n, n-1, \ldots, 1)$. Thus our solution for $RF2|\ell = 3|C_{\max}$ under Condition (4.2) is a new solution for a larger class than the one solved in Wang, Sethi, and van de Velde (1997).

Given a partition job k, where $k \in \mathcal{N}$, and an integer time point s, where $0 \leq s \leq a$, it is clear that *if* we know the left-jobs, then the right-jobs are also known (as the complementary set to the left-jobs), and the whole schedule can be determined. Thus, to find the optimal schedule, it suffices to consider only the different possibilities

for the left-jobs. Therefore, for $j = 1, \ldots, k-1, k+1, \ldots, n$, we will implement a subroutine to generate the best (partial) schedule for the set of jobs $\{1, 2, \ldots, j\}\backslash\{k\}$ such that the total length of the 1-operations, 2-operations, and 3-operations of the left-jobs is t_1, t_2, and t_3, respectively.

Let $F_j^{k,s}(t_1, t_2, t_3)$ denote the *maximum* slack of the *right-jobs* for any schedule for the set of jobs $\{1, 2, \ldots, j\}\backslash\{k\}$, $j = 1, \ldots, k-1, k+1, \ldots, n$, such that the 1-operations, 2-operations, and 3-operations of the left-jobs fill up the time intervals $[0, t_1]$, $[s, s+t_2]$, and $[a+b, a+b+t_3]$, respectively, see Figure 4.7. It is clear that the schedule corresponding to $F_n^{k,s}(t_1, t_2, t_3)$ is the best schedule of $\mathcal{N} \setminus \{k\}$ for some given t_1, t_2, t_3, k, and s values. We now derive recursive relations to calculate $F_j^{k,s}(t_1, t_2, t_3)$. Since job j can be scheduled either as a left-job or as a right-job, we have two scenarios.

First, consider job j as a left-job in a (partial) schedule ψ'. We can obtain this schedule by inserting job j as the last left-job into the partial schedule $\overline{\psi}$ associated with $F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j})$. See Figures 4.8 and 4.9. Clearly, the insertion

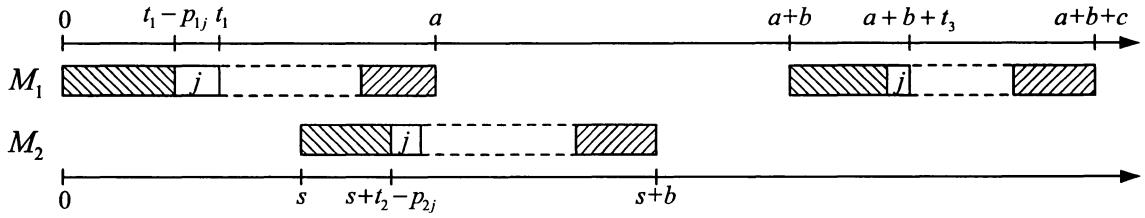

Figure 4.8: Job j as a left-job in a schedule ψ' when $\phi = (n, n-1, \ldots, 1)$

of job j is possible only if $S_{2j}(\psi') \geq C_{1j}(\psi')$, i.e., $s + t_2 - p_{2j} \geq t_1$; otherwise, the 1-operation and 2-operation of job j would overlap. Since the 2-operation of a left-job is always completed no later than time point a, it is clear that any left-job has a slack of b time units. Now let us consider the slack of the right-jobs in ψ'. We discuss separately the aforementioned two cases of ϕ:

If $\phi = (n, n-1, \ldots, 1)$, then all previously scheduled right-jobs were located in the reverse direction starting from the right end of their block, and thus the slack of the right-jobs in ψ' is the same as in $\overline{\psi}$ because their schedule remains intact. See Figure 4.8. Therefore, the slack of the right-jobs in a feasible schedule ψ' is given by $\Delta_r(\psi') = F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j})$. Thus,

$$\Delta_r(\psi') = \begin{cases} F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}), & \text{if } s + t_2 \geq t_1 + p_{2j}; \\ -\infty, & \text{otherwise,} \end{cases}$$

where $\Delta_r(\psi') = -\infty$ means that the schedule ψ' is infeasible.

If $\phi = (1, 2, \ldots, n)$, then all previously scheduled right-jobs were located in increasing order of their index starting immediately to the right of the partition job

k, and thus the 1-operations, 2-operations, and 3-operations of these right-jobs will start p_{1j}, p_{2j}, and p_{3j} units of time later in ψ' than in $\overline{\psi}$, respectively, when job j is inserted. See Figure 4.9. In this case, the slack of the right-jobs in ψ' may increase by

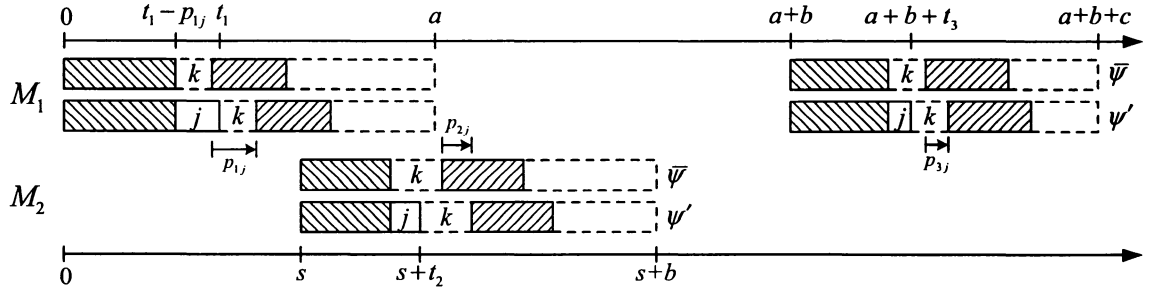

Figure 4.9: Job j as a left-job in a schedule ψ' when $\phi = (1, 2, \ldots, n)$

$p_{3j} - p_{2j}$ if $p_{3j} > p_{2j}$ or decrease by $p_{2j} - p_{3j}$ if $p_{3j} < p_{2j}$ compared to what it was in $\overline{\psi}$. (Note that the slack of a right-job j'' in a feasible schedule ψ' is always positive because $S_{3j''}(\psi') \geq a+b+t_3$ and $C_{2j''}(\psi') \leq s+b$.) Therefore, the slack of the right-jobs in a feasible schedule ψ' is given by $\Delta_r(\psi') = F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}) + p_{3j} - p_{2j}$ if $F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}) > 0$; otherwise, $\Delta_r(\psi') = F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j})$. (Here $F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}) \leq 0$ implies that the schedule $\overline{\psi}$ is infeasible or there is no right-job in a feasible $\overline{\psi}$.) Thus,

$$\Delta_r(\psi') = \begin{cases} F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}) + p_{3j} - p_{2j}, \\ \qquad \text{if } s + t_2 \geq t_1 + p_{2j} \text{ and } F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}) > 0; \\ F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}), \\ \qquad \text{if } s + t_2 \geq t_1 + p_{2j} \text{ and } F_{j-1}^{k,s}(t_1 - p_{1j}, t_2 - p_{2j}, t_3 - p_{3j}) \leq 0; \\ -\infty, \text{ if } s + t_2 < t_1 + p_{2j}, \end{cases}$$

where $\Delta_r(\psi') = -\infty$ means that the schedule ψ' is infeasible.

Second, consider job j as a right-job in a (partial) schedule ψ''. We can obtain this schedule by adding job j to the partial schedule associated with $F_{j-1}^{k,s}(t_1, t_2, t_3)$. Since a right-job always starts its 2-operation later than time point a, it is clear that job j automatically satisfies the feasibility condition in the schedule ψ''. Let $a(j) = \sum_{i=1, i\neq k}^{j} p_{1i}$, $b(j) = \sum_{i=1, i\neq k}^{j} p_{2i}$, and $c(j) = \sum_{i=1, i\neq k}^{j} p_{3i}$, where $1 \leq j \leq n$. To calculate the slack of job j in ψ'', we consider the two cases of ϕ again:

If $\phi = (n, n-1, \ldots, 1)$, then the 1-operation, 2-operation, and 3-operation of job j in ψ'' is scheduled in the time intervals $[a - a(j) + t_1,\ a - a(j-1) + t_1]$, $[s+b-b(j)+t_2,\ s+b-b(j-1)+t_2]$, and $[a+b+c-c(j)+t_3,\ a+b+c-c(j-1)+t_3]$, respectively. See schedule ψ_1'' in Figure 4.10. Furthermore, the given ϕ-order implies that any other right-jobs, which have a smaller index than j, are placed to the right of job j in ψ_1''. It is easy to see that the slack of job j in ψ'' is equal to $\delta_j(\psi'') = S_{3j}(\psi'') -$

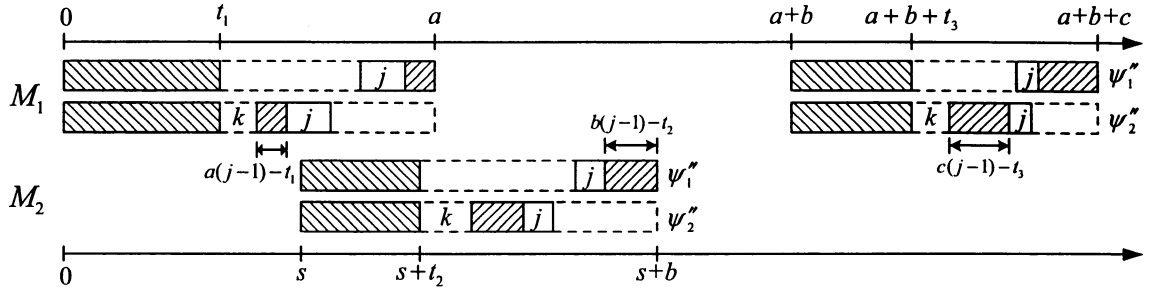

Figure 4.10: Job j as a right-job in a schedule ψ''

$C_{2j}(\psi'') = a+b+c-c(j)+t_3-(s+b-b(j-1)+t_2)$. Therefore, the slack of the right-jobs in ψ'' is given by $\Delta_r(\psi'') = \min\{F_{j-1}^{k,s}(t_1,t_2,t_3),\ a+c+t_3+b(j-1)-s-t_2-c(j)\}$ if $F_{j-1}^{k,s}(t_1,t_2,t_3) \neq 0$; otherwise, $\Delta_r(\psi'') = a+c+t_3+b(j-1)-s-t_2-c(j)$. (When $F_{j-1}^{k,s}(t_1,t_2,t_3) = 0$, there is no right-job in the feasible, partial schedule associated with $F_{j-1}^{k,s}(t_1,t_2,t_3)$.)

If $\phi = (1,2,\ldots,n)$, then the 1-operation, 2-operation, and 3-operation of job j in ψ'' is scheduled in the time intervals $[p_{1k}+a(j-1),\ p_{1k}+a(j)]$, $[s+p_{2k}+b(j-1),\ s+p_{2k}+b(j)]$, and $[a+b+p_{3k}+c(j-1),\ a+b+p_{3k}+c(j)]$, respectively. See schedule ψ_2'' in Figure 4.10. Furthermore, the current ϕ-order implies that any other right-jobs, which have a smaller index than j, are placed to the left of job j in ψ_2''. In this case, the slack of job j in ψ'' is equal to $\delta_j(\psi'') = S_{3j}(\psi'') - C_{2j}(\psi'') = a+b+p_{3k}+c(j-1)-(s+p_{2k}+b(j))$. Therefore, for this case the slack of the right-jobs in ψ'' is given by $\Delta_r(\psi'') = \min\{F_{j-1}^{k,s}(t_1,t_2,t_3),\ a+b+p_{3k}+c(j-1)-s-p_{2k}-b(j)\}$ if $F_{j-1}^{k,s}(t_1,t_2,t_3) \neq 0$; otherwise, $\Delta_r(\psi'') = a+b+p_{3k}+c(j-1)-s-p_{2k}-b(j)$.

As $F_j^{k,s}(t_1,t_2,t_3)$ is the larger of the slack of the right-jobs in ψ' and ψ'', we have $F_j^{k,s}(t_1,t_2,t_3) = \max\{\Delta_r(\psi'),\Delta_r(\psi'')\}$. The initialization of the recursion can be set by

$$F_0^{k,s}(t_1,t_2,t_3) = \begin{cases} 0, & \text{if } t_1=t_2=t_3=0; \\ -\infty, & \text{otherwise} \end{cases}$$

for $1 \leq k \leq n$, $0 \leq s \leq a$, $0 \leq t_1 \leq a$, $0 \leq t_2 \leq b$, and $0 \leq t_3 \leq c$. Moreover, if $j = k$, we let $F_j^{k,s}(t_1,t_2,t_3) = F_{j-1}^{k,s}(t_1,t_2,t_3)$.

Finally, we have to take into account the possible effect of the partition job k on the slack, as well as on the feasibility of the schedule. For this, consider now the partition job k in the schedule $\psi_n^{k,s}(t_1,t_2,t_3)$ associated with $F_n^{k,s}(t_1,t_2,t_3)$. We schedule k immediately after the left-jobs. More precisely, we put the 1-operation, 2-operation, and 3-operation of job k in the time intervals $[t_1,t_1+p_{1k}]$, $[s+t_2,s+t_2+p_{2k}]$, and $[a+b+t_3,a+b+t_3+p_{3k}]$, respectively. See Figure 4.7. This is possible only if

a. $S_{2k}(\psi_n^{k,s}(t_1,t_2,t_3)) \geq C_{1k}(\psi_n^{k,s}(t_1,t_2,t_3))$, i.e., $s+t_2 \geq t_1+p_{1k}$; otherwise, the 1-operation and 2-operation of job k would overlap; and

b. $s + t_2 \leq a$ and $s + t_2 + p_{2k} > a$; otherwise, job k cannot be the partition job.

Let $G^{k,s}(t_1, t_2, t_3)$ denote the slack of the right-jobs and the partition job in the schedule $\psi_n^{k,s}(t_1, t_2, t_3)$. As $S_{3k}(\psi_n^{k,s}(t_1, t_2, t_3)) - C_{2k}(\psi_n^{k,s}(t_1, t_2, t_3)) = (a+b+t_3) - (s+t_2+p_{2k})$, we have

$$G^{k,s}(t_1, t_2, t_3) = \begin{cases} \min\{F_n^{k,s}(t_1, t_2, t_3),\ (a+b+t_3) - (s+t_2+p_{2k})\}, \\ \qquad \text{if } s + t_2 \geq t_1 + p_{1k} \text{ and } a - s - p_{2k} < t_2 \leq a - s; \\ -\infty, \text{ otherwise,} \end{cases}$$

where $G^{k,s}(t_1, t_2, t_3) = -\infty$ means that the schedule $\psi_n^{k,s}(t_1, t_2, t_3)$ is infeasible.

The overall maximum slack $\Delta_r(\psi^*)$ for right-jobs can be found as

$$\Delta_r(\psi^*) = \max_{1 \leq k \leq n,\ 0 \leq s \leq a,\ 0 \leq t_1 \leq a,\ 0 \leq t_2 \leq b,\ 0 \leq t_3 \leq c} G^{k,s}(t_1, t_2, t_3).$$

The minimum makespan is then given by $C_{\max}(\psi^*) = a + b + c - \min\{b, \Delta_r(\psi^*)\}$. The optimal partition $(\mathcal{N}_1^*, k^*, \mathcal{N}_2^*)$ of $\mathcal{N}$ can be retrieved by backtracking. Finally, schedule the jobs in $\mathcal{N}_1^*$ in increasing order of the indexes, followed by job k^*, followed by the jobs in $\mathcal{N}_2^*$ in ϕ-order to obtain the optimal schedule ψ^*.

The worst case computation time and space required by the above algorithm can be determined as follows. There are at most $O(nabc)$ states for the functions $F_j^{k,s}(t_1, t_2, t_3)$ with fixed k and s. There are at most n partition jobs k and a time points s. Therefore, there are at most $O(n^2a^2bc)$ recursive equations to be solved in the dynamic programming algorithm. As the calculation for each recursive equation takes constant time, the overall running time of the algorithm is bounded by $O(n^2a^2bc)$, which is clearly pseudo-polynomial. Since only recursive functions take up space and there are at most $O(nabc)$ states for the functions $F_j^{k,s}(t_1, t_2, t_3)$ with fixed k and s, it is easy to see that the algorithm requires $O(nabc)$ space. Thus we have proved the following theorem.

Theorem 4.5 *There exists a dynamic programming algorithm that solves the problem* $RF2|\ell = 3|C_{\max}$ *under Condition (4.2) or (4.3) in* $O(n^6p_{\max}^4)$ *time and* $O(n^4p_{\max}^3)$ *space, where* $p_{\max} = \max_{1 \leq i \leq 3, 1 \leq j \leq n} p_{ij}$.

Unfortunately, the above dynamic programming recursion does not work for arbitrary ϕ-orders, since when adding a job j as a right-job, it can be positioned anywhere among the previously placed (scheduled) right-jobs from $\{1, 2, \ldots, j-1\}$ in a schedule. Our solution uses the fact that ϕ is a monotone order. This approach could be further extended to the case when ϕ consists of a fixed number r of monotone segments, for example to "pyramidal" ϕ-orders corresponding to $r = 2$.

Returning to the $J2|m_j \leq 3|C_t$ problem, if Condition (4.2) or (4.3) is satisfied by $\mathcal{J}_{212}$, then the best sequence for $\mathcal{J}_{212}$ and its corresponding schedule σ' can be

produced by the above dynamic programming algorithm, which takes $O(n^6 p_{\max}^4)$ time and $O(n^4 p_{\max}^3)$ space. After the partial schedule σ' for $\mathcal{J}_{212}$ has been built, the job insertion can be performed in $O(n)$ time and constant space. Thus, the optimal schedule σ^* for $J2|m_j \leq 3|C_t$ can be found in $O(n^6 p_{\max}^4)$ time and $O(n^4 p_{\max}^3)$ space. We therefore have the following result.

Corollary 4.6 *The problem $J2|m_j \leq 3|C_t$ under Condition (4.2) or (4.3) for the jobs in $\mathcal{J}_{212}$ can be solved in $O(n^6 p_{\max}^4)$ time and $O(n^4 p_{\max}^3)$ space, where $p_{\max} = \max_{1\leq i\leq 3, 1\leq j\leq n} p_{ij}$.*

4.3.2 FPTAS

Consider an $\mathcal{NP}$-hard cost-minimization problem Π. An algorithm $\mathcal{A}$ is said to be a *fully polynomial time approximation scheme* (FPTAS) for Π if on input (I, ϵ), where I is any instance of Π and $\epsilon > 0$ is an arbitrary error bound, it always returns a solution whose cost is at most $(1+\epsilon)$ times the optimal cost and its running time is bounded by a polynomial in the size of instance I and $1/\epsilon$. Under the assumption $\mathcal{P} \neq \mathcal{NP}$, an FPTAS is the best one can hope for an $\mathcal{NP}$-hard optimization problem (see e.g., Vazirani (2001) for explanations). Due to the fact that the existence of FPTASs is intimately related to the existence of pseudo-polynomial time algorithms, the natural question arises whether it is true for the $RF2|\ell = 3|C_{\max}$ problem. In this section we give a positive answer to this question.

A pseudo-polynomial time algorithm displays exponential behavior only when an input instance includes exponentially large numbers. Thus, to obtain an FPTAS for our problem, we exploit the idea of rounding down the input so that the numbers become polynomial in n and then apply the pseudo-polynomial time algorithm. This rounding incurs some loss of precision, but we will show that the resulting minimum makespan is at most $(1+\epsilon)$ times the optimal makespan.

Suppose there exists a pseudo-polynomial time algorithm, which will be called *Pseudo-RF2* hereafter, for the general or special cases of the $RF2|\ell = 3|C_{\max}$ problem. Let $\lfloor x \rfloor$ denote the largest integer that does not exceed x. An FPTAS is described as follows.

Algorithm FPTAS-RF2
Input: p_{ij} $(i = 1, 2, 3$ and $j = 1, 2, \ldots, n)$ and $\epsilon > 0$.
1. Let $a = \sum_{j=1}^{n} p_{1j}$, $b = \sum_{j=1}^{n} p_{2j}$, $c = \sum_{j=1}^{n} p_{3j}$, and $K = \epsilon \max\{a + c, b\}/(n+2)$.
2. For $i = 1, 2, 3$ and $j = 1, 2, \ldots, n$, define $\widehat{p}_{ij} = \lfloor p_{ij}/K \rfloor$.
3. Using these modified processing times as input, apply Algorithm Pseudo-RF2 to the modified problem to find its optimal processing sequence $\widehat{\sigma}$.

Output: A no-passing block schedule for the original problem in which jobs are processed according to $\widehat{\sigma}$.

In what follows, we show that Algorithm FPTAS-RF2 is indeed an FPTAS for $RF2|\ell = 3|C_{\max}$. Without loss of generality, we assume that the jobs are re-numbered in such a way that $\widehat{\sigma} = (1, 2, ..., n)$. Moreover, we assume $C_{\max}(\widehat{\sigma}) \neq a + c$; otherwise, the schedule $\widehat{\sigma}$ is optimal for the original problem as well. Suppose that jobs $\widehat{u}$ and $\widehat{v}$ are critical in schedule $\widehat{\sigma}$, i.e., $C_{\max}(\widehat{\sigma}) = \sum_{j=1}^{\widehat{u}} p_{1j} + \sum_{j=\widehat{u}}^{\widehat{v}} p_{2j} + \sum_{j=\widehat{v}}^{n} p_{3j}$. As $K\widehat{p}_{ij} \leq p_{ij} \leq K\widehat{p}_{ij} + K$ for $i = 1, 2, 3$ and $j = 1, 2, \ldots, n$, we derive

$$\begin{aligned} C_{\max}(\widehat{\sigma}) &= \sum_{j=1}^{\widehat{u}} p_{1j} + \sum_{j=\widehat{u}}^{\widehat{v}} p_{2j} + \sum_{j=\widehat{v}}^{n} p_{3j} \\ &\leq \sum_{j=1}^{\widehat{u}} (K\widehat{p}_{1j} + K) + \sum_{j=\widehat{u}}^{\widehat{v}} (K\widehat{p}_{2j} + K) + \sum_{j=\widehat{v}}^{n} (K\widehat{p}_{3j} + K) \\ &= K \left(\sum_{j=1}^{\widehat{u}} \widehat{p}_{1j} + \sum_{j=\widehat{u}}^{\widehat{v}} \widehat{p}_{2j} + \sum_{j=\widehat{v}}^{n} \widehat{p}_{3j} \right) + (n+2)K. \end{aligned}$$

Since for the optimal schedule σ^*, we deduce from (4.1) that

$$\begin{aligned} C_{\max}(\sigma^*) &= \max \left\{ a + c, \max_{1 \leq \mu, \nu \leq n} \left\{ \sum_{j=1}^{\mu} p_{1\sigma^*(j)} + \sum_{j=\mu}^{\nu} p_{2\sigma^*(j)} + \sum_{j=\nu}^{n} p_{3\sigma^*(j)} \right\} \right\} \\ &\geq K \max_{1 \leq \mu, \nu \leq n} \left\{ \sum_{j=1}^{\mu} \widehat{p}_{1\sigma^*(j)} + \sum_{j=\mu}^{\nu} \widehat{p}_{2\sigma^*(j)} + \sum_{j=\nu}^{n} \widehat{p}_{3\sigma^*(j)} \right\} \\ &\geq K \left(\sum_{j=1}^{\widehat{u}} \widehat{p}_{1j} + \sum_{j=\widehat{u}}^{\widehat{v}} \widehat{p}_{2j} + \sum_{j=\widehat{v}}^{n} \widehat{p}_{3j} \right). \end{aligned}$$

The last inequality is obtained from the fact that $\widehat{\sigma}$ is optimal for the modified problem.

Combining the above two formulas, we get $C_{\max}(\widehat{\sigma}) \leq C_{\max}(\sigma^*) + (n+2)K = C_{\max}(\sigma^*) + \epsilon \max\{a+c, b\}$. Then using the observation that $C_{\max}(\sigma^*) \geq \max\{a+c, b\}$, yields the desired upper bound $C_{\max}(\widehat{\sigma}) \leq (1+\epsilon) C_{\max}(\sigma^*)$.

It is not difficult to see that the running time of Algorithm FPTAS-RF2 is bounded by a polynomial in the input size and $1/\epsilon$. As an example, let us take a look at the pseudo-polynomial time algorithm presented in Section 4.3.1. Since the running time of the dynamic programming algorithm on the modified problem is $O(n^2\widehat{a}^2\widehat{b}\widehat{c})$, the running time of Algorithm FPTAS-RF2 is not more than $O(n^2a^2bc/K^4) = O(n^6/\epsilon^4)$, which is polynomial in n and $1/\epsilon$. Also, it is easy to find that this algorithm requires $O(n\widehat{a}\widehat{b}\widehat{c}) \leq O(nabc/K^3) = O(n^4/\epsilon^3)$ space. We thus have the following theorem.

Theorem 4.7 *If there exists a pseudo-polynomial time algorithm for the* $RF2|\ell = 3|C_{\max}$ *problem, it admits an FPTAS too.*

4.3.3 Well-solvable cases

If we consider only no-passing block schedules, the $RF2|\ell = 3|C_{\max}$ problem is equivalent to the three-machine flow shop problem—denoted by $F3||C_{\max}$—with the additional constraint that no 3-operation can start until all jobs finish their 1-operation. For a given instance of $RF2|\ell = 3|C_{\max}$, we first solve it as an $F3||C_{\max}$ problem. Let π_F^* denote an optimal (block) schedule for the $F3||C_{\max}$ problem. If $C_{\max}(\pi_F^*) > a + c$, then the schedule π_F^* is also feasible and optimal for $RF2|\ell = 3|C_{\max}$ because $\max\{C_{\max}(\pi_F^*), a + c\}$ is a lower bound on the minimum makespan. If $C_{\max}(\pi_F^*) \leq a + c$, we can easily turn π_F^* into a feasible schedule π_{RF} for $RF2|\ell = 3|C_{\max}$ by delaying the start of 3-operations at the time when all the 1-operations are completed. In this case, $C_{\max}(\pi_{RF}) = a + c$ and π_{RF} is optimal too. Relying on such reduction, Wang, Sethi, and van de Velde (1997) have identified the following special cases of $RF2|\ell = 3|C_{\max}$ that are solvable in $O(n \log n)$ time.

Case 1: One machine dominates another in terms of processing times, i.e., one of the following conditions holds:

1. $\min_j\{p_{1j}\} \geq \max_j\{p_{2j}\}$.
2. $\min_j\{p_{3j}\} \geq \max_j\{p_{2j}\}$.
3. $\min_j\{p_{2j}\} \geq \max_j\{p_{1j}\}$.
4. $\min_j\{p_{2j}\} \geq \max_j\{p_{3j}\}$.

Case 2: Recessive second stage: $p_{2j} \leq \min\{p_{1j}, p_{3j}\}$ for all j.

Case 3: Constant second stage: $p_{2j} = constant$.

Using Theorem 4.2, this leads to the following corollary.

Corollary 4.8 *The problem $J2|m_j \leq 3|C_t$ under any one of the above conditions for the jobs in $\mathcal{J}_{212}$ can be solved in $O(n \log n)$ time.*

Aside from the listed cases, there are additional polynomially solvable cases for $F3||C_{\max}$. On the basis of the above discussion and Theorem 4.2, if the condition for those special cases is satisfied by the jobs in $\mathcal{J}_{212}$, then $J2|m_j \leq 3|C_t$ is solvable in polynomial time as well.

4.4 Summary

We have studied the cycle time minimization problem in a two-machine job shop, where the maximum number of operations of any job is bounded by three. We reduced the problem to the makespan minimization problem in a two-machine reentrant flow shop. Based on previously known and newly extended results for the reentrant flow shop problem, we devised exact and approximate solutions for special cases of the job shop problem. We also presented an approximate solution for the

general case of the problem, and recognized a few polynomially solvable cases. The exact complexity status of the general problem remains an open question for future research.

Chapter 5
The Traveling Salesman Problem

5.1 Introduction

The *traveling salesman problem* (TSP) is one of the most widely studied problems in combinatorial optimization. Simply, the problem may be stated as follows: Given a collection of "cities," find the shortest tour that visits all of them exactly once and returns to the starting city.

The TSP belongs to the class of difficult optimization problems, as it is strongly $\mathcal{NP}$-hard. Nevertheless, many special cases of it can be solved in polynomial time when the distance matrix satisfies certain properties. For a comprehensive review of the extensive results on this subject, the interested reader is referred to the earlier survey by Gilmore, Lawler, and Shmoys (1985), as well as the recent papers by Burkard et al. (1998) and Kabadi (2002).

The TSP plays an important role in applications like production scheduling. A large number of scheduling problems can be formulated as special cases of the TSP, and many well-solvable cases of the TSP originate from scheduling problems. For example, we know that a special case of the TSP with distance matrix $C = (c_{ij}) = \max\{f_i, e_j\}$ can be solved by the Gilmore–Gomory algorithm (Gilmore and Gomory 1964), which was originally designed to solve the problem of sequencing jobs on a one-state-variable machine. This also can be used to find the minimum makespan in the two-machine no-wait flow shop scheduling problem in $O(n \log n)$ time (Reddi and Ramamoorthy 1972). Thus investigating special cases of the TSP that can be solved by polynomial algorithms is of great practical significance.

An $n \times n$ matrix $C = (c_{ij})$ is a *Monge (distribution) matrix*, if it fulfills the so-called Monge property:

$$c_{ij} + c_{i'j'} \leq c_{ij'} + c_{i'j} \text{ for all } 1 \leq i < i' \leq n \text{ and } 1 \leq j < j' \leq n.$$

Furthermore, $C = (c_{ij})$ is a *permuted Monge (distribution) matrix*, if there exists a permutation ϕ such that $C^{\phi} = (c_{i\phi(j)})$ is a Monge matrix. These matrices can be recognized and their permutation ϕ can be found in $O(n^2)$ time (Burkard et al. 1998). The Monge property has received considerable attention in combinatorial optimization (Burkard, Klinz, and Rudolf 1996), as its particular structure often leads to easier solutions for problems. For instance, for the linear assignment problem with

a Monge cost matrix, the *identity permutation* $\phi(i) = i$ for $i = 1, 2, \ldots, n$ is an optimal solution. Based on this fact, if ϕ is the permutation for which C^{ϕ} is a Monge matrix, then ϕ is an optimal assignment for C. See e.g., Gilmore, Lawler, and Shmoys (1985) for details.

For the TSP with a Monge distance matrix, there exists an optimal tour which is pyramidal. If we number the cities by $1, 2, \ldots, n$, then a tour is called *pyramidal*, if starting from the initial city 1, they are first visited in increasing order of their index, and then the remaining cities are visited in decreasing order. For example, the six-city tour $1 \to 3 \to 5 \to 6 \to 4 \to 2 \to 1$ is pyramidal, but the tour $1 \to 5 \to 3 \to 6 \to 4 \to 2 \to 1$ is not. For any distance matrix $C = (c_{ij})$, a shortest pyramidal tour can be found by an efficient dynamic programming scheme in $O(n^2)$ time: Let $Q(i, j)$ denote the length of a shortest pyramidal path from city i to city j that visits every city in $\{1, 2, \ldots, \max\{i, j\}\}$ exactly once. Here, a path is said to be *pyramidal*, if it first passes through the cities in descending order of index from i to 1 and then in ascending order of index from 1 to j. By decomposing a pyramidal path into smaller parts, it is not difficult to see that

$$Q(j, j+1) = \min_{1 \le i < j} \left\{ Q(i+1, i) + c_{i,j+1} + \sum_{k=i+1}^{j-1} c_{k+1,k} \right\}$$

and

$$Q(j+1, j) = \min_{1 \le i < j} \left\{ Q(i, i+1) + c_{j+1,i} + \sum_{k=i+1}^{j-1} c_{k,k+1} \right\}.$$

Note that $\sum_{k=i+1}^{j-1} c_{k+1,k} = \sum_{k=i+1}^{j-1} c_{k,k+1} = 0$ if $i \ge j - 1$. Starting from the initial conditions $Q(1, 2) = c_{12}$ and $Q(2, 1) = c_{21}$, this recurrence allows us to compute $Q(i, j)$ for all $1 \le i, j \le n$ and $i \ne j$. For example, the recurrence yields

$$\begin{aligned}
&Q(2,3) = Q(2,1) + c_{13}, \quad Q(3,2) = Q(1,2) + c_{31}, \\
&Q(3,4) = \min\{Q(2,1) + c_{14} + c_{32},\ Q(3,2) + c_{24}\}, \\
&Q(4,3) = \min\{Q(1,2) + c_{41} + c_{23},\ Q(2,3) + c_{42}\}, \\
&Q(4,5) = \min\{Q(2,1) + c_{15} + c_{32} + c_{43},\ Q(3,2) + c_{25} + c_{43},\ Q(4,3) + c_{35}\}, \\
&Q(5,4) = \min\{Q(1,2) + c_{51} + c_{23} + c_{34},\ Q(2,3) + c_{52} + c_{34},\ Q(3,4) + c_{53}\}
\end{aligned}$$

for $n = 5$. The length of a shortest pyramidal tour τ_n on cities $1, 2, \ldots, n$ is then given by

$$c(\tau_n) = \min\{Q(n-1, n) + c_{n,n-1},\ Q(n, n-1) + c_{n-1,n}\}.$$

See e.g., Gilmore, Lawler, and Shmoys (1985) for details. Further improvement in the time complexity can be achieved if the distance matrix C is a Monge matrix. By exploiting the combinatorial structure of Monge matrices, Park (1991) showed that the calculation can be speeded up so that the TSP on Monge matrices is solvable in $O(n)$ time.

A matrix $C = (c_{ij})$ of the form $c_{ij} = a_i b_j$ with real numbers a_i and b_j, $1 \leq i, j \leq n$, is called a *product matrix*. Sarvanov (1980) proved that the TSP on product matrices is $\mathcal{NP}$-hard. Since product matrices are contained in the more general class of permuted Monge matrices, it follows immediately that the TSP with a permuted Monge matrix is also $\mathcal{NP}$-hard.

In this chapter, we investigate special cases of the TSP on permuted Monge matrices. Our work is related to cases of the TSP studied in the context of scheduling a two-machine bufferless *robotic cell*, where a robot is used (as a material handling device) to load, unload, and move parts between machines, see Sethi et al. (1992), Hall, Kamoun, and Sriskandarajah (1997), Aneja and Kamoun (1999), Crama et al. (2000), Middendorf and Timkovsky (2002), as well as Chapters 2 and 3. An optimal solution of this scheduling problem can be found by solving an n-city TSP with distance matrix $C = (c_{ij}) = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$, where μ, b_i' and a_j's are given non-negative numbers. By using the Gilmore–Gomory algorithm as a subroutine on auxiliary problems, Aneja and Kamoun (1999) described an $O(n \log n)$ time algorithm for this TSP. Our study of a robotic scheduling problem dealing with K-component parts gave rise to a more general version of the TSP with distance matrix $C = (c_{ij}) = \min\{\lambda + \eta + Kb_i + a_j, \max\{\lambda + (K-1)b_i + \max\{\eta + \rho, b_i\}, a_j\}\}$, where all parameters are given non-negative numbers again (cf. Chapter 3). This matrix was shown to belong to the class of permuted Monge matrices in Chapter 3. Notice also that when $K = 1$ and $\lambda = \eta = 0$, then this last distance matrix reduces to the matrix of Aneja and Kamoun (1999).

In this chapter, we define a hierarchy of new classes of permuted Monge distance matrices. Using the theory of subtour patching, we give new, polynomial-time solutions for the corresponding cases of the TSP. We also discuss efficient recognition algorithms for these matrices. Our algorithms also solve the K-component robotic scheduling problem introduced in Chapter 3 and its special case—the problem studied in Aneja and Kamoun (1999). Our most general algorithm has $O(n^2)$ complexity, which can be improved to $O(n \log n)$ in some special cases.

The remainder of the chapter is organized as follows. In Section 5.2, we introduce several definitions and review the theory of subtour patching. In Section 5.3, we derive our main results for both the general and the special cases. We demonstrate our new algorithm on an example in the chapter. Section 5.4 concludes the chapter with our final remarks.

5.2 Preliminaries

5.2.1 Permutations

For any n-city TSP with a given $n \times n$ distance matrix $C = (c_{ij})$, we denote the set of cities by $\{1, 2, \ldots, n\}$. A *permutation* ϕ on these n elements, which may be

written as

$$\phi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \phi(1) & \phi(2) & \cdots & \phi(n) \end{pmatrix},$$

corresponds to an assignment $i \to \phi(i)$ for $i = 1, 2, \ldots, n$ with the associated cost $c(\phi) = \sum_{i=1}^{n} c_{i\phi(i)}$. An *optimal assignment* is one that minimizes the cost, and can be computed in polynomial time. An assignment can also be expressed as a directed graph $G = (V, E)$ with vertices $V = \{1, 2, \ldots, n\}$ and arcs $E = \{(i, \phi(i)) : 1 \leq i \leq n\}$, representing the fact that $\phi(i)$ is visited immediately after city i. It is easy to see that every vertex of G has both in-degree and out-degree exactly one. (Given a directed graph, the *in-degree of a vertex* is the number of arcs that point to it and the *out-degree of a vertex* is the number of arcs that point away from it.) If the associated graph G is connected—there is at least one path from every vertex to every other vertex, then ϕ forms a (TSP) tour, and ϕ is said to be a *cyclic permutation.* Otherwise, ϕ consists of several subtours (or cycles).

Often, an assignment (i.e., permutation) ϕ is stated in compact form in terms of factors. Let $i_1, i_2, \ldots, i_q$ be distinct elements of $\{1, 2, \ldots, n\}$. If $\phi(i_k) = i_{k+1}$ for $k = 1, 2, \ldots, q-1$, and $\phi(i_q) = i_1$, then $(i_1, i_2, \ldots, i_q)$ is called a *factor* (or *cycle*, or *subtour*) of the assignment ϕ. A factor with $q = 1$ (in this case $\phi(i_1) = i_1$) will be called a *trivial factor.* For example, the assignment

$$\phi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 2 & 5 & 6 & 1 & 4 \end{pmatrix} = (1,3,5)(2)(4,6)$$

has three factors (or subtours) $\phi_1 = (1, 3, 5)$, $\phi_2 = (2)$, and $\phi_3 = (4, 6)$. Among these three, ϕ_2 is a trivial factor.

We can modify an assignment ϕ by multiplying it with a permutation ψ, which produces a new assignment ϕ' defined as $\phi'(i) = \phi \circ \psi(i) = \phi(\psi(i))$ for $i = 1, 2, \ldots, n$. A *transposition* $\langle i, j \rangle$ is a permutation that interchanges i and j. An *adjacent transposition* is of the form $\langle i, i+1 \rangle$. Performing $\langle i, j \rangle$ on the permutation ϕ yields the permutation $\phi' = \phi \circ \langle i, j \rangle$ with $\phi'(i) = \phi(j)$, $\phi'(j) = \phi(i)$, and $\phi'(k) = \phi(k)$ for $k \neq i, j$. Recall that the product of two transpositions is a non-commutative operation if they have a common index, otherwise it is commutative. For a given permutation ϕ, the *cost of performing transposition* $\langle i, j \rangle$ *on* ϕ is defined by $c_\phi(i, j) = c(\phi \circ \langle i, j \rangle) - c(\phi) = c_{i\phi(j)} + c_{j\phi(i)} - c_{i\phi(i)} - c_{j\phi(j)}$.

5.2.2 Review of the theory of subtour patching

As our solution technique is based on the theory of *subtour patching*, we briefly review its important points that will be used in this chapter; the interested reader is referred to Gilmore, Lawler, and Shmoys (1985) and Burkard et al. (1998) for a more comprehensive coverage.

In general, the strategy of subtour patching works as follows: First solve an assignment problem for the given distance matrix. If the optimal assignment ϕ is a

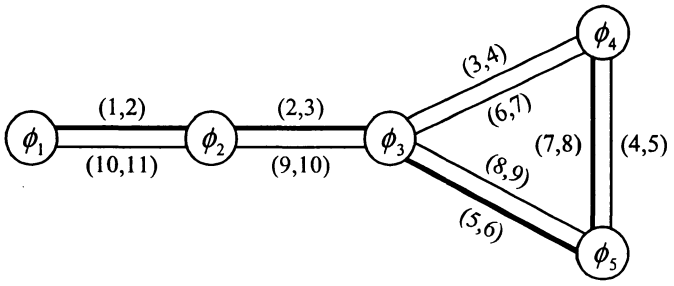

Figure 5.1: A patching graph $G_\phi = (V, E)$

tour, it is clearly optimal for the underlying TSP, as the cost of ϕ is a lower bound on the length of an optimal tour. Otherwise, the assignment ϕ consists of $r \geq 2$ subtours $\phi_1, \phi_2, \ldots, \phi_r$, i.e., $\phi = \phi_1\phi_2 \cdots \phi_r$. In this case, patch the subtours together by a series of transpositions so as to yield an optimal solution for the TSP. More precisely, if i and j are in different subtours of ϕ, then performing the transposition $\langle i, j \rangle$ on ϕ will patch these two subtours into a single tour. A series of transpositions, in any order, can be expressed as a single permutation by forming their product. For an optimal assignment ϕ, a permutation ψ is called a *patching permutation* if $\phi \circ \psi$ is a cyclic permutation (tour). Thus the problem is "Given an optimal assignment ϕ, find an *optimal patching permutation* ψ^* such that $\phi \circ \psi^*$ is an optimal tour."

In order to determine an optimal patching permutation, it is often useful to examine the so-called *patching graph.* With respect to a given optimal assignment ϕ, a patching graph $G_\phi = (V, E)$ may be constructed as follows: The vertices are the subtours ϕ_i of ϕ, $1 \leq i \leq r$. Every edge $e \in E$ corresponds to an adjacent transposition $\langle i, i+1 \rangle$, that is, if city i is in subtour ϕ_j and city $i+1$ is in subtour ϕ_k, $j \neq k$, then the two corresponding vertices in G_ϕ are connected by an edge labeled $(i, i+1)$. For the sake of simplicity, we will refer to edge $(i, i+1)$ as $\bar{i}$. Since the same pair of vertices may be connected by multiple edges, this construction will yield a connected *multigraph* with r vertices and at most $n-1$ edges. As an example, suppose we have an 11-city TSP with the optimal assignment $\phi = \phi_1\phi_2\phi_3\phi_4\phi_5 = (1,11)(2,10)(3,6,9)(4,7)(5,8)$. The corresponding patching graph $G_\phi = (V, E)$ is shown in Figure 5.1, where $V = \{\phi_i : 1 \leq i \leq 5\}$ and $E = \{(i, i+1) : 1 \leq i \leq 10\}$.

A *spanning tree* T of a graph G is a tree connecting all its vertices. A permutation obtained by multiplying a set of adjacent transpositions which correspond to a spanning tree in G_ϕ is called a *tree permutation.* Gilmore and Gomory (1964) have shown that every tree permutation is a patching permutation. For example, the edges $\bar{1}$, $\bar{2}$, $\bar{5}$, and $\bar{7}$ form a spanning tree in G_ϕ of the above example (see the bold lines of Figure 5.1). Thus the product of the transpositions $\langle 1, 2 \rangle$, $\langle 2, 3 \rangle$, $\langle 5, 6 \rangle$, and $\langle 7, 8 \rangle$, in *any* order, forms a tree permutation, which patches together the subtours of ϕ into a tour. With each edge $\bar{i}$ in G_ϕ, we associate a non-negative *weight* w_i that represents the cost of performing the corresponding transposition $\langle i, i+1 \rangle$ on ϕ, i.e.,

$w_i = c(\phi \circ \langle i, i+1 \rangle) - c(\phi) = c_\phi(i, i+1)$. The *weight* $w(T)$ *of a tree* T is defined as the sum of the weights of the edges in T. A *minimum-weight spanning tree*, or *MST* for short, is a spanning tree of G whose weight is minimum.

Recall that if a permutation (assignment) ψ consists of t subtours $\psi_1, \psi_2, \ldots, \psi_t$, then

$$c(\phi \circ \psi) - c(\phi) = \sum_{i=1}^{t} (c(\phi \circ \psi_i) - c(\phi)), \tag{5.1}$$

see Theorem 14 in Gilmore, Lawler, and Shmoys (1985). Corresponding to a given spanning tree T, we define ψ_T as a patching permutation that minimizes (5.1). An $n \times n$ matrix $C = (c_{ij})$ will be called a *Gilmore–Gomory matrix* if it is defined by

$$C = (c_{ij}) = \begin{cases} \int_{\alpha_i}^{\beta_j} g_1(x)dx & \text{if } \beta_j \geq \alpha_i; \\ \int_{\beta_j}^{\alpha_i} g_2(x)dx & \text{if } \beta_j < \alpha_i, \end{cases} \tag{5.2}$$

where the given functions g_1 and g_2 are integrable, and $g_1(x) + g_2(x) \geq 0$ for all x. (Note that from the above assumption, functions g_1 and g_2 need not be strictly non-negative.) To solve a special case of the TSP with distance matrix (5.2), Gilmore and Gomory (1964) developed a subtour-patching strategy that uses only adjacent transpositions with minimum total cost, which can be found by determining an MST for the patching graph. Burdyuk and Trofimov (1976) and Gilmore, Lawler, and Shmoys (1985) proved that this basic patching strategy of using only adjacent transpositions is extendible also to permuted Monge matrices. Their result is essentially contained in the following theorem.

Theorem 5.1 (Burdyuk and Trofimov 1976; Gilmore, Lawler, and Shmoys 1985) *Let* $C^\phi = (c_{i\phi(j)})$ *be a Monge matrix. For any cyclic permutation* π*, there exists a spanning tree* $T = \{\overline{i_1}, \overline{i_2}, \ldots, \overline{i_{r-1}}\}$ *of the patching graph* G_ϕ *and a sequence* σ *for performing the transpositions of* T *such that the permutation* $\phi_T = \phi \circ \langle i_{\sigma(1)}, i_{\sigma(1)} + 1 \rangle \circ \langle i_{\sigma(2)}, i_{\sigma(2)} + 1 \rangle \circ \cdots \circ \langle i_{\sigma(r-1)}, i_{\sigma(r-1)} + 1 \rangle$ *is a cyclic permutation with* $c(\phi_T) \leq c(\pi)$.

Theorem 5.1 is important as it allows us to restrict our search for an optimal patching permutation only to those permutations which can be formed of adjacent transpositions of G_ϕ, but it *does not* say anything about how to find the spanning tree T corresponding to these transpositions and the sequence σ in which they have to be multiplied. In the following we take a closer look at these problems. A set of edges in G_ϕ is said to be *dense* if it is of the form $\{\overline{i}, \overline{i+1}, \ldots, \overline{j}\}$ with $j \geq i$. Let T be a spanning tree for the patching graph G_ϕ. We partition the set of edges (i.e., the transpositions) of T into t $(1 \leq t \leq r-1)$ dense, pairwise disjoint subsets $\mathcal{I}(i_1, j_1) = \{\overline{i_1}, \overline{i_1+1}, \ldots, \overline{j_1}\}, \mathcal{I}(i_2, j_2) = \{\overline{i_2}, \overline{i_2+1}, \ldots, \overline{j_2}\}, \ldots, \mathcal{I}(i_t, j_t) = \{\overline{i_t}, \overline{i_t+1}, \ldots, \overline{j_t}\}$, which will be called *branches* hereafter, such that $T = \mathcal{I}(i_1, j_1) \cup \mathcal{I}(i_2, j_2) \cup \cdots \cup \mathcal{I}(i_t, j_t)$ and $j_k + 1 < i_{k+1}$ for $k = 1, 2, \ldots, t-1$. Refer to the example illustrated in Figure

5.1. Suppose again that $T = \{\overline{1}, \overline{2}, \overline{5}, \overline{7}\}$. Then T is composed of three branches with $\mathcal{I}(1,2) = \{\overline{1}, \overline{2}\}$, $\mathcal{I}(5,5) = \{\overline{5}\}$, and $\mathcal{I}(7,7) = \{\overline{7}\}$. Since $j_k + 1 < i_{k+1}$ for any two branches $\mathcal{I}(i_k, j_k)$ and $\mathcal{I}(i_{k+1}, j_{k+1})$, $k = 1, 2, \ldots, t-1$, the permutations corresponding to transpositions of different branches have no common element and thus are commutative and can be performed independent of each other in any order. Furthermore, since the patching costs of these permutations are additive by Equation (5.1), the best patching permutation corresponding to T can be identified by finding the minimum cost permutation for each branch and taking the product of these. It is well known that, as branches are dense, performing transpositions of a branch $\mathcal{I}(i,j)$ in *any* order will yield a pyramidal tour on the set of cities $\{i, i+1, \ldots, j, j+1\}$. Hence, the best patching permutation ψ_T can be derived by constructing a shortest pyramidal tour on the set of cities $\{i_k, i_k+1, \ldots, j_k, j_k+1\}$ for each branch $\mathcal{I}(i_k, j_k)$ of T, where $1 \leq k \leq t$. Since finding the shortest pyramidal tour on a given set of cities is solvable in polynomial time, the remaining hard part of the problem is how to find the best spanning tree.

Let us now define the *b-weight* w^b_{ij} *of a branch* $\mathcal{I}(i,j)$ by

$$w^b_{ij} = c(\phi \circ \psi^*_i) - c(\phi),$$

where ψ^*_i is a shortest pyramidal subtour corresponding to the branch $\mathcal{I}(i,j)$. Also, we define the *weight* w_{ij} *of a branch* $\mathcal{I}(i,j)$ as the total weight of the edges in the branch, i.e., $w_{ij} = \sum_{k=i}^{j} w_k$. Note that from these definitions, we have $w^b_{ii} = w_{ii} = w_i = c_\phi(i, i+1)$ for a branch $\mathcal{I}(i,i)$ that contains only edge $\overline{i}$. Further, we define the *b-weight* $w^b(T)$ *of a spanning tree* T as the total b-weight of its branches, i.e., $w^b(T) = \sum_{k=1}^{t} w^b_{i_k,j_k}$. It is easy to verify from the definitions of ψ_T and $w^b(T)$ that $w^b(T) = c(\phi \circ \psi_T) - c(\phi)$. As $c(\phi)$ is constant, ψ_T is an optimal patching permutation *if and only if* the corresponding spanning tree T has minimum b-weight. In other words, the best spanning tree T^* is actually a *minimum-b-weight spanning tree* of G_ϕ. Thus, the TSP on permuted Monge matrices is essentially reduced to the problem of finding a spanning tree of G_ϕ with minimum b-weight.

If the distance matrix C is a permuted Monge matrix, then it has been established by Burkard et al. (1998) that for any branch $\mathcal{I}(i,j)$, we have $w^b_{ij} \geq w^b_{ik} + w^b_{k+1,j}$ for $1 \leq i \leq n-1$ and $i \leq k < j \leq n-1$, and $w^b_{ij} \geq \sum_{k=i}^{j} w_k (= w_{ij})$ for $1 \leq i \leq j \leq n$, i.e., the b-weight is *super-additive*. This further implies that for a spanning tree T its b-weight is never lower than its weight, i.e., $w^b(T) \geq w(T)$. A permuted Monge matrix is said to be a *b-weight-additive matrix* if the b-weight of any branch is additive, i.e., $w^b_{ij} = \sum_{k=i}^{j} w_k$ for $1 \leq i \leq j \leq n$. It is important to note that these matrices can be recognized in $O(n^2)$ time combining Park's (1991) method for computing the b-weights with the algorithm for recognizing permuted Monge matrices. From the definition of an MST and Theorem 5.1, we know that the cost of the optimal assignment ϕ plus the weight of the MST $\widehat{T}$ for G_ϕ, i.e., $c(\phi) + w(\widehat{T})$, is a lower bound on the length of an optimal tour, which is equal to $c(\phi) + w^b(T^*)$ as explained above.

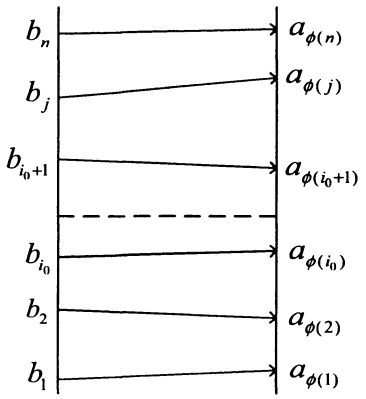

Figure 5.2: The assignment ϕ for $C = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$

Consequently, if the subtours of ϕ can be patched with cost $w(\widehat{T})$, the resulting tour is clearly optimal, and we are done. Since for b-weight-additive matrices, the b-weight of a spanning tree is the same as its weight, the MST is the best patching tree in this case. It can be shown that the b-weight-additivity property is satisfied by Gilmore–Gomory matrices (Burkard et al. 1998). Hence, Gilmore–Gomory matrices form a subclass of the b-weight-additive matrices.

5.3 Polynomially Solvable Classes

5.3.1 b-decomposable matrices

Consider an $n \times n$ permuted Monge matrix $C = (c_{ij})$ such that $D = (d_{ij})$ with $d_{ij} = c_{i\phi(j)}$ is a Monge matrix. We call C *b-decomposable* if D can be partitioned by an index i_0 $(1 \leq i_0 \leq n)$ into two b-weight-additive sub-matrices $D' = (d'_{ij})$ for $i, j \leq i_0$ and $D'' = (d''_{ij})$ for $i, j > i_0$. Note that if $i_0 = n$, then D'' is empty and D itself is a b-weight-additive matrix. We show in this section that the class of the TSP with a b-decomposable distance matrix is solvable in polynomial time.

As we have already noted, this special case of the TSP originated from robotic-cell scheduling problems. Let us take a look, for example, at the aforementioned matrix $C = (c_{ij}) = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$, where μ is a given constant and all numbers are non-negative. Without loss of generality, we assume that the cities have been renumbered so that $b_1 \leq \cdots \leq b_n$. Let ϕ be an assignment (ordering) for which $a_{\phi(i)} \leq a_{\phi(i+1)}$. See Figure 5.2 for an illustration. It can be shown that ϕ is an assignment for which $C^\phi = (c_{i\phi(j)})$ is a Monge matrix (Aneja and Kamoun 1999). Observe that $b_i + a_{\phi(i)} \leq b_{i+1} + a_{\phi(i+1)}$ for all $i = 1, 2, \ldots, n-1$. If there exists an index i_0, $i_0 < n$, such that $b_{i_0} + a_{\phi(i_0)} \leq \mu < b_{i_0+1} + a_{\phi(i_0+1)}$, then the Monge matrix $D = C^\phi = (c_{i\phi(j)})$ can be split into two sub-matrices $D' = (d'_{ij})$ and $D'' = (d''_{ij})$, where D' is a *sum matrix*, i.e., $d'_{ij} = b_i + a_{\phi(j)}$ for $i, j \leq i_0$, and

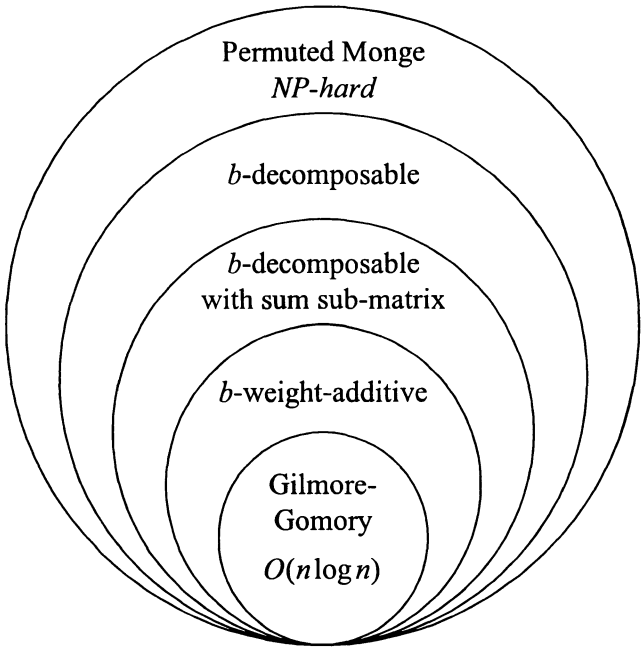

Figure 5.3: Hierarchical classes of permuted Monge matrices

$d''_{ij} = \max\{\mu, b_i, a_{\phi(j)}\}$ for $i, j > i_0$ is a Gilmore–Gomory matrix. For a sum matrix, it is b-weight-additive. (In fact, the b-weight of any branch is always zero because all (cyclic) permutations on the n cities have the same cost.) Thus matrix C is b-decomposable. It was shown in Chapter 3 that the matrix of the robotic scheduling problem with K-component parts, as mentioned in Section 5.1 (defined in Section 3.3), is also b-decomposable. Actually, both scheduling problems lead to matrices belonging to the *subclass* of b-decomposable matrices in which D' is a sum matrix. Of course, the class of b-decomposable matrices is substantially larger, as a matrix in it can have *any* b-weight-additive sub-matrix for its two parts. Figure 5.3 depicts a hierarchy of these newly defined matrix classes.

Let us return now to b-decomposable TSP-s in general. Since ϕ is the permutation for which $C^\phi = (c_{i\phi(j)})$ is a Monge matrix, ϕ is an optimal assignment for C, and ϕ is also optimal for the TSP in case it is a tour. As a result, hereafter, we concentrate on the case when ϕ consists of $r \geq 2$ subtours. Let $G_\phi = (V, E)$ be the patching graph relative to ϕ. We will use the subtour-patching technique by reformulating our TSP as a minimum-b-weight spanning tree problem.

For a given spanning tree T of G_ϕ, we begin with a characterization of the branches of T. Let $\mathcal{I}(i, j)$ be a branch of T. If $\mathcal{I}(i, j)$ does not contain the edge $\overline{i_0}$, then the weights of the whole edge set of $\mathcal{I}(i, j)$ are either in D' or in D'', and thus $w^b_{ij} = w_{ij}$, because both D' and D'' are b-weight-additive. Otherwise, it is known that $w^b_{ij} \geq w_{ij}$. As we will demonstrate later, the branch containing the edge $\overline{i_0}$, if it exists, is of primary interest among all branches of T, as it is the *only* branch which may

Table 5.1: The Example Instance

cities	1	2	3	4	5	6	7	8	9	10
a_i	100	120	10	32	130	90	30	110	20	39
b_i	5	15	25	36	45	83	95	106	117	125
$a_{\phi(i)}$	10	20	30	32	39	90	100	110	120	130
$\phi(i)$	3	9	7	4	10	6	1	8	2	5

not be b-weight-additive. From here on, we will refer to this branch as a *b-branch.*

Let us examine now specifically the MST $\widehat{T}$ of G_ϕ. Note that during the construction of $\widehat{T}$ by Kruskal's (1956) algorithm (see e.g., Ahuja, Magnanti, and Orlin 1993), if there is an edge available with the same weight as edge $\overline{i_0}$, we consider that edge first. This manner of construction will ensure that $\widehat{T}$ does not have a b-branch unless it is necessary. If $\widehat{T}$ does not contain a b-branch, then the b-weight of $\widehat{T}$ is the same as the weight of the tree, and the TSP is solved by using this MST as a patching tree. If $\widehat{T}$ contains a b-branch, then it must be of the form $\mathcal{I}(h_0, j_0) = \{\overline{h_0}, \ldots, \overline{i_0 - 1}, \overline{i_0}, \overline{i_0 + 1}, \ldots, \overline{j_0}\}$ for some $h_0 \leq i_0$ and $j_0 \geq i_0$. Then the b-weight of $\widehat{T}$ is calculated as the b-weight $w^b_{h_0,j_0}$ of the branch $\mathcal{I}(h_0, j_0)$ plus the weights of all edges in $\widehat{T} \backslash \mathcal{I}(h_0, j_0)$. If $\mathcal{I}(h_0, j_0) = \{\overline{i_0}\}$, then $w^b_{h_0,j_0} = w_{h_0,j_0} = w_{i_0}$, implying that $w^b(\widehat{T}) = w(\widehat{T})$, which means $\widehat{T}$ is a minimum-b-weight spanning tree. Thus we assume for the remainder of the discussion that $|\mathcal{I}(h_0, j_0)| > 1$. Clearly, $w^b_{h_0,j_0}$ can be obtained by using an algorithm for finding a shortest pyramidal tour on the set of cities $\{h_0, \ldots, i_0 - 1, i_0, i_0 + 1, \ldots, j_0, j_0 + 1\}$ with the corresponding distances extracted from the matrix $D = (d_{ij}) = (c_{i\phi(j)})$. Since D is a Monge matrix, this can be done in linear time by using Park's (1991) recursions. Note that in Park's recursions, by computing the length of a shortest pyramidal tour on the set of cities $\{1, 2, \ldots, n\}$, we can obtain at the same time the lengths of the shortest pyramidal tours for all sets of cities $\{1, 2, \ldots, j\}$ for $j = 2, \ldots, n$. Therefore, these recursions allow us to calculate in linear time all b-weights $w^b_{h_0,h_0+1}, w^b_{h_0,h_0+2}, \ldots, w^b_{h_0,j_0}$. Now let us compare the weight w_{h_0,j_0} of the branch $\mathcal{I}(h_0, j_0)$ with its b-weight. If $w^b_{h_0,j_0} = w_{h_0,j_0}$, then the b-weight of $\widehat{T}$ is the same as the weight of the tree, and the problem is solved by using $\widehat{T}$ as a patching tree. Otherwise, i.e., in the case when $w^b_{h_0,j_0} > w_{h_0,j_0}$, the MST $\widehat{T}$ may not be the best spanning tree to patch the subtours of ϕ, as illustrated by the following example.

Example Let us consider a 10-city TSP with the distance matrix $C = (c_{ij}) = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$, where $\mu = 80$. The b's and a's are given in Table 5.1. Observe that the b_i-s are in the order $b_i \leq b_{i+1}$. Table 5.1 also shows the ordering ϕ for which $a_{\phi(i)} \leq a_{\phi(i+1)}$.

As discussed above, $C^\phi = (c_{i\phi(j)})$ is a Monge matrix and thus ϕ is an optimal assignment for C. Furthermore, it is easy to see that ϕ consists of six subtours

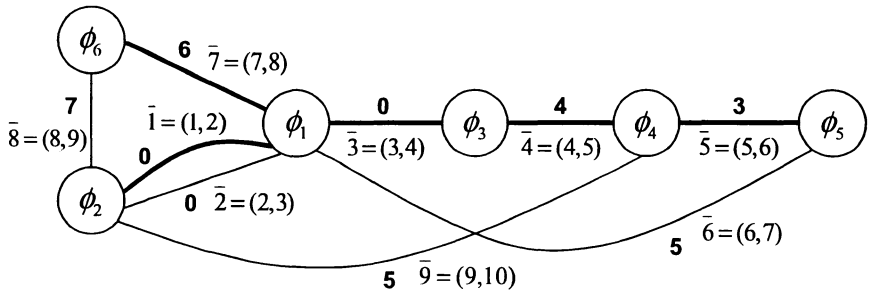

Figure 5.4: The patching graph G_ϕ in the example

$\phi = \phi_1\phi_2\phi_3\phi_4\phi_5\phi_6 = (1,3,7)(2,9)(4)(5,10)(6)(8)$, and the cost of ϕ is equal to $c(\phi) = \sum_{i=1}^{n} c_{i\phi(i)} = 803$. The patching graph G_ϕ, as shown in Figure 5.4, has 6 vertices and 9 edges. (In Figure 5.4, the numbers in bold are edge-weights.)

Since $b_4 + a_{\phi(4)} < \mu$ and $b_5 + a_{\phi(5)} > \mu$, we have $i_0 = 4$, which means that $D' = (d'_{ij}) = b_i + a_{\phi(j)}$ for all $i, j \leq 4$, and $D'' = (d''_{ij}) = \max\{\mu, b_i, a_{\phi(j)}\}$ for all $i, j > 4$. The weights of the edges as determined by $w_i = c_{i\phi(i+1)} + c_{i+1,\phi(i)} - c_{i\phi(i)} - c_{i+1,\phi(i+1)}$ are as follows: $w_1 = w_2 = w_3 = 0$, $w_4 = 4$, $w_5 = 3$, $w_6 = 5$, $w_7 = 6$, $w_8 = 7$, and $w_9 = 5$. It is easy to see that $\widehat{T} = \{\bar{1}, \bar{3}, \bar{4}, \bar{5}, \bar{7}\}$ is an MST in G_ϕ, and it is indicated in bold line in Figure 5.4. The spanning tree contains three branches $\mathcal{I}(1,1)$, $\mathcal{I}(3,5)$ and $\mathcal{I}(7,7)$. According to the patching scheme, we have to calculate the b-weight of the tree, which is the sum of the three branches' b-weights. The b-weight of $\mathcal{I}(1,1)$ and $\mathcal{I}(7,7)$ is 0 and 6, respectively. Now let us consider the set of cities $\{3,4,5,6\}$ with the corresponding distances extracted from the matrix $D = (d_{ij}) = (c_{i\phi(j)})$. Using the recursions introduced in Section 5.1, we can obtain

$$
\begin{aligned}
&Q(1,2) = 57, \quad Q(2,1) = 66,\\
&Q(2,3) = 66 + 64 = 130, \quad Q(3,2) = 57 + 75 = 132,\\
&Q(3,4) = \min\{66 + 90 + 77,\ 132 + 90\} = 222,\\
&Q(4,3) = \min\{57 + 83 + 75,\ 130 + 83\} = 213.
\end{aligned}
$$

Then we have $c(\tau_3) = \min\{130 + 77,\ 132 + 75\} = 207$ and $c(\tau_4) = \min\{222 + 83,\ 213 + 90\} = 303$. By backtracking, a shortest pyramidal tour on cities $3,4,5,6$ is $\tau_4 = (3,5,6,4)$. The b-weight of branches $\mathcal{I}(3,4)$ and $\mathcal{I}(3,5)$ can thus be determined as $w^b_{3,4} = c(\tau_3) - \sum_{i=3}^{5} c_{i\phi(i)} = 207 - 55 - 68 - 80 = 4$ and $w^b_{3,5} = c(\tau_4) - \sum_{i=3}^{6} c_{i\phi(i)} = 303 - 55 - 68 - 80 - 90 = 10$, respectively. Unfortunately, the inequality $w^b_{3,5} > w_3 + w_4 + w_5 = 7$ holds for this branch. In this case, another spanning tree may have a lower b-weight. For instance, the spanning tree $T' = \{\bar{2}, \bar{3}, \bar{5}, \bar{6}, \bar{8}\}$ has a b-weight of 15, which is lower than $w^b(\widehat{T}) = w_1 + w^b_{3,5} + w_7 = 16$. (Since T' contains no edge $\overline{i_0}$, its b-weight can easily be calculated as $w^b(T') = w(T') = 15$.)

The above example demonstrates that the MST $\widehat{T}$ may not have minimum b-weight if $w^b_{h_0,j_0} > w_{h_0,j_0}$. Thus we need to determine a minimum-b-weight spanning

tree T^* for this case, which may contain a b-branch. Of course, if we knew which of the b-branches has to be included in the patching tree, the other edges could easily be found by any of the greedy algorithms for finding an MST, such as Kruskal's (1956) algorithm. Hence, our focus here is on the selection of the b-branch. A straightforward strategy would be to search through all possible b-branches. Before we present our more efficient search procedure, we note that we cannot restrict the choices of the b-branch only to sub-branches of $\mathcal{I}(h_0, j_0)$—the b-branch of the MST $\widehat{T}$. For example, if we decide to remove edge $\overline{h_0}$ from the tree $\widehat{T}$ (and from the branch $\mathcal{I}(h_0, j_0)$), a new edge has to be added to form a spanning tree in the patching graph. If the edge selected happens to be the edge $\overline{j_0+1}$, its addition will create a new branch $\mathcal{I}'(h_0+1, j_x)$ with $j_x > j_0$. For instance, in the example above, if we remove edge $\overline{3}$ from $\widehat{T}$ and add edge $\overline{6}$, a new branch $\mathcal{I}'(4,7) = \{\overline{4}, \overline{5}, \overline{6}, \overline{7}\}$ will be created.

A b-branch must start with an edge $\overline{h}$ with $1 \le h \le i_0$. We call such a branch a *b_h-branch*. Let $h_{\min}$ be the minimal possible index of the starting edge of a b-branch in the patching graph G_ϕ. The b-branches can then be chosen exclusively from among b_h-branches with $h_{\min} \le h \le i_0$. It is clear that $h_{\min}$ can be easily determined as the smallest index h ($h \ge 1$) for which $\{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}\}$ is a cycle-free path in G_ϕ. For the example above, since it will create a cycle $1 \to 2 \to 1$ when edge $\overline{1}$ is inserted, we have $h_{\min} = 2$.

We are now ready to state our search strategy: For each h, $h_{\min} \le h \le i_0$, first find a minimum-b-weight spanning tree T_h^b in $G_\phi \backslash \{\overline{h-1}\}$ among all trees containing a b_h-branch (we define $G_\phi \backslash \{\overline{0}\} = G_\phi$). After this, find the minimum-b-weight spanning tree T^b among the T_h^b-s.

Next we give details of a linear-time procedure to find the tree T_h^b. This means that the tree T^b can be obtained in $O(n^2)$ time. To find an MST when the underlying graph changes, we use the following lemma from Ahuja, Magnanti, and Orlin (1993).

Lemma 5.2 *Let T be an MST for a graph G. If an edge (i,j) of T is removed from G, which results in a graph $G' = G \backslash \{(i,j)\}$ and two components of T containing sets of vertices V' and V'', then $T' = T \cup \{(i',j')\} \setminus \{(i,j)\}$ is an MST for G', where (i',j') is an edge with the minimum weight in the cut $[V', V'']$ of G'.*

Proof. The proof follows from the "cut optimality conditions" for a minimum-weight spanning tree. That is, for every edge (i,j) of T', its weight is not larger than the weight of any edge (k,l) contained in the cut of G' formed by deleting edge (i,j) from T'. See e.g., Theorem 13.1 in Ahuja, Magnanti, and Orlin (1993, p. 518) for details. □

Given h, $h_{\min} \le h \le i_0$, we first find, if possible, a spanning tree $\widehat{T}_h$ in $G'_\phi = G_\phi \backslash \{\overline{h-1}\}$ with minimum weight $w(\widehat{T}_h)$ among all trees containing the edges $\overline{h}, \overline{h+1}, \ldots, \overline{i_0}$: Start with the initial working sub-tree $\{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}\}$, then $\widehat{T}_h$ can be determined by Kruskal's MST algorithm, which keeps adding to the tree the next

smallest-weight edge from those that remain as long as it does not create a cycle. It should be noted here that the removal of edge $\overline{h-1}$ from G_ϕ may disconnect the patching graph, which would make finding a spanning tree in G'_ϕ impossible. Therefore, if $\widehat{T}_h$ does not exist, then there is no T_h^b either; otherwise, let $\mathcal{I}(h, j_h) = \{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}, \overline{i_0+1}, \ldots, \overline{j_h}\}$ be the b-branch in $\widehat{T}_h$. Then the b-weight of $\widehat{T}_h$ can be calculated as the b-weight of the branch $\mathcal{I}(h, j_h)$ plus the weights of all edges in $\widehat{T}_h \backslash \mathcal{I}(h, j_h)$, i.e., $w^b(\widehat{T}_h) = w^b_{h,j_h} + w(\widehat{T}_h) - \sum_{k=h}^{j_h} w_k$. For spanning trees of G_ϕ which contain a b_h-branch, there is a simple but very useful property as shown in the following lemma.

Lemma 5.3 *For any spanning tree T of G_ϕ that contains a branch $\mathcal{I}(h, j)$ such that $j \geq j_h$, its b-weight is not less than that of $\widehat{T}_h$, i.e., $w^b(T) \geq w^b(\widehat{T}_h)$.*

Proof. Notice that both $\widehat{T}_h$ and T have the edges $\overline{h}, \overline{h+1}, \ldots, \overline{i_0}, \overline{i_0+1}, \ldots, \overline{j_h}$. Since $\widehat{T}_h$ has minimum weight among all spanning trees containing a b_h-branch, the total weight of the remaining edges of T is not less than the same in $\widehat{T}_h$—that is, $w(\widehat{T}_h) - \sum_{k=h}^{j_h} w_k$. Moreover, as the b-weight fulfills the property $w^b_{ij} \geq w^b_{ik} + w^b_{k+1,j}$ for any $i \leq k < j$, it is easy to verify that $w^b(T) \geq w^b_{h,j_h} + w(\widehat{T}_h) - \sum_{k=h}^{j_h} w_k = w^b(\widehat{T}_h)$. □

Lemma 5.3 tells us that we can restrict our search for the tree T_h^b only to $\widehat{T}_h$ and trees containing sub-branches of $\mathcal{I}(h, j_h)$. We will take advantage of this observation in our search. If the b-weight of the branch $\mathcal{I}(h, j_h)$ is the same as its weight, i.e., $w^b_{h,j_h} = w_{h,j_h}$, then $T_h^b = \widehat{T}_h$; otherwise, we assume without loss of generality that $j_h \geq i_0 + 1$. The search for T_h^b can be conducted by computing next, one by one, the b-weights for the MST-s containing branches $\mathcal{I}(h, j-1)$, $j = i_0 + 1, i_0 + 2, \ldots, j_h$. Notice that if we remove, for instance, edge $\overline{j}$ for some $j \in [i_0 + 1, j_h]$ from $\widehat{T}_h$, then based on Lemma 5.2, the MST that contains the branch $\mathcal{I}(h, j-1) = \{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}, \overline{i_0+1}, \ldots, \overline{j-1}\}$, if it exists, can be found by connecting two components of $\widehat{T}_h \backslash \{\overline{j}\}$ by the edge $\overline{s(j)}$ with the minimum weight $w_{s(j)}$ in the cut between the two components of $\widehat{T}_h \backslash \{\overline{j}\}$. Furthermore, it is easy to verify that the b-weight of this spanning tree is equal to $w(\widehat{T}_h) - \sum_{k=h}^{j} w_k + w_{s(j)} + w^b_{h,j-1}$. As we already mentioned above, all b-weights $w^b_{h,i_0}, w^b_{h,i_0+1}, \ldots, w^b_{h,j_h}$ can be calculated in linear time. At this point, the only question left to be answered is how to find in linear time the "replacement" edges $\overline{s(j)}$ for all j with $i_0 + 1 \leq j \leq j_h$.

In order to identify the replacement edges, we represent the tree $\widehat{T}_h$ as a rooted tree with the vertex (subtour) containing index $i_0 + 1$ as the *root*. For each vertex u in the tree $\widehat{T}_h$, we assign a *pointer* $p(u)$ to be the largest index of the edges from $\mathcal{I}(h, j_h)$ on the path from the root to vertex u. For the vertices for which the path from the root does not contain any edge from $\mathcal{I}(h, j_h)$ at all, we define $p(u) = i_0$. For the root,

we assign its pointer to be i_0. Now consider an edge $\bar{i}$ of $G_\phi \backslash \{\overline{h-1}\}$, which is not in the tree $\widehat{T}_h$. Suppose it connects vertices v_1 and v_2 with $p(v_1) = h_1$ and $p(v_2) = h_2$, $h_1 \leq h_2$. If $h_1 < h_2$, then this edge can be used as a replacement edge after removing any one of the edges $\overline{h_1+1}, \ldots, \overline{h_2}$; otherwise, i.e. when $h_1 = h_2$, it cannot be used as a replacement for any edge $\bar{j}$ with $i_0 + 1 \leq j \leq j_h$ at all, as it will not be in any cut between the two components of $\widehat{T}_h \backslash \{\bar{j}\}$. Therefore, by looking through the sorted list of edges (e.g., in non-decreasing order of weight) not included in $\widehat{T}_h$, we can identify a best replacement edge, if possible, for each edge $\bar{j}$ with $i_0 + 1 \leq j \leq j_h$. Thus clearly, the entire replacement search can be executed in linear time.

Let $h = 4$ in the example above (see also Figure 5.5). It is easy to verify that an MST of $G_\phi \backslash \{\overline{h-1}\} = G_\phi \backslash \{\overline{3}\}$ that contains a b_h-branch is $\widehat{T}_h = \widehat{T}_4 = \{\overline{1}, \overline{4}, \overline{5}, \overline{6}, \overline{7}\}$. So $\mathcal{I}(h, j_h) = \mathcal{I}(4, 7)$. Note that edge $\overline{3}$ has been removed from the patching graph G_ϕ. The pointers p are defined as follows: $p(\phi_4) = 4$, $p(\phi_5) = 5$, $p(\phi_1) = 6$, $p(\phi_6) = 7$, $p(\phi_2) = 6$, and $p(\phi_3) = 4$. Here edges $\overline{2}$, $\overline{8}$ and $\overline{9}$ are not in the tree. For edge $\overline{2}$, as $p(\phi_1) = p(\phi_2)$, it cannot be used as replacement for any of the edges $\overline{5}$, $\overline{6}$ and $\overline{7}$. For edge $\overline{8}$, as $p(\phi_2) = 6$ and $p(\phi_6) = 7$, it could be a replacement for edge $\overline{7}$. For edge $\overline{9}$, as $p(\phi_4) = 4$ and $p(\phi_2) = 6$, it could be a replacement for edges $\overline{5}$ and $\overline{6}$. Now let us determine the minimum-b-weight spanning tree T_4^b in $G_\phi \backslash \{\overline{3}\}$ among all trees containing a b_4-branch. It is clear that the b-weight of branch $\mathcal{I}(4, 4)$ is equal to $w_{4,4}^b = w_4 = 4$. Using the dynamic programming scheme, the b-weight of branches $\mathcal{I}(4, 5)$, $\mathcal{I}(4, 6)$, and $\mathcal{I}(4, 7)$ can be computed as follows: Consider the set of cities $\{4, 5, 6, 7, 8\}$ with the corresponding distances extracted from the matrix $D = (d_{ij}) = (c_{i\phi(j)})$. Compute the recursions

$$
\begin{aligned}
&Q(1,2) = 75, \quad Q(2,1) = 77,\\
&Q(2,3) = 77 + 90 = 167, \quad Q(3,2) = 75 + 83 = 158,\\
&Q(3,4) = \min\{77 + 100 + 83,\ 158 + 100\} = 258,\\
&Q(4,3) = \min\{75 + 95 + 90,\ 167 + 95\} = 260,\\
&Q(4,5) = \min\{77 + 110 + 83 + 95,\ 158 + 110 + 95,\ 260 + 110\} = 363,\\
&Q(5,4) = \min\{75 + 106 + 90 + 100,\ 167 + 106 + 100,\ 258 + 106\} = 364.
\end{aligned}
$$

Then we obtain $c(\tau_3) = \min\{167 + 83,\ 158 + 90\} = 248$, $c(\tau_4) = \min\{258 + 95,\ 260 + 100\} = 353$, and $c(\tau_5) = \min\{363 + 106,\ 364 + 110\} = 469$. This gives us $w_{4,5}^b = c(\tau_3) - \sum_{i=4}^{6} c_{i\phi(i)} = 248 - 238 = 10$, $w_{4,6}^b = c(\tau_4) - \sum_{i=4}^{7} c_{i\phi(i)} = 353 - 338 = 15$, and $w_{4,7}^b = c(\tau_5) - \sum_{i=4}^{8} c_{i\phi(i)} = 469 - 448 = 21$. Therefore, the tree $\widehat{T}_4$ has a b-weight of $w^b(\widehat{T}_4) = w_1 + w_{4,7}^b = 21$. It is easy to verify that the MST-s containing branches $\mathcal{I}(4, 4)$, $\mathcal{I}(4, 5)$ and $\mathcal{I}(4, 6)$ are $\{\overline{1}, \overline{4}, \overline{6}, \overline{7}, \overline{9}\}$, $\{\overline{1}, \overline{4}, \overline{5}, \overline{7}, \overline{9}\}$ and $\{\overline{1}, \overline{4}, \overline{5}, \overline{6}, \overline{8}\}$, respectively. Furthermore, the b-weight of these trees is 20, 21 and 22, respectively. Hence, we have $T_4^b = \{\overline{1}, \overline{4}, \overline{6}, \overline{7}, \overline{9}\}$ and $w^b(T_4^b) = 20$.

Similarly, we can find that $\widehat{T}_3 = \{\overline{1}, \overline{3}, \overline{4}, \overline{5}, \overline{7}\}$ in $G_\phi \backslash \{\overline{2}\}$. As already shown, the computation of the b-weight for branch $\mathcal{I}(3, 5)$ gives $w_{3,4}^b = 4$ and $w_{3,5}^b = 10$.

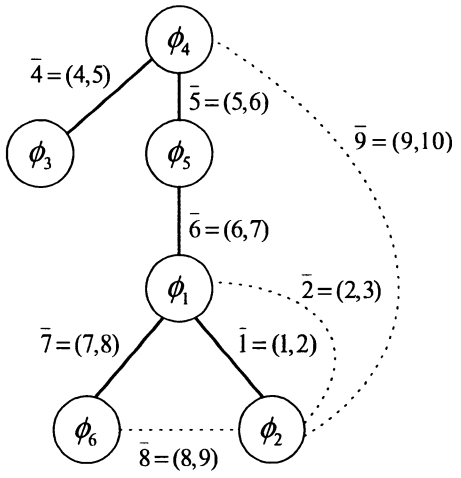

Figure 5.5: A rooted tree for edge replacement

Replacing edge $\bar{5}$ with edge $\bar{6}$ in $\widehat{T}_3$, we obtain the tree $\{\bar{1},\bar{3},\bar{4},\bar{6},\bar{7}\}$. Its b-weight is equal to $w_1 + w^b_{3,4} + w_6 + w_7 = 15$, which is lower than $w^b(\widehat{T}_3) = w_1 + w^b_{3,5} + w_7 = 16$. This yields $T^b_3 = \{\bar{1},\bar{3},\bar{4},\bar{6},\bar{7}\}$ and $w^b(T^b_3) = 15$. Again, beginning from $\widehat{T}_2 = \{\bar{2},\bar{3},\bar{4},\bar{5},\bar{7}\}$ in $G_\phi \backslash \{\bar{1}\}$, we can find that $T^b_2 = \{\bar{2},\bar{3},\bar{4},\bar{6},\bar{7}\}$ and $w^b(T^b_2) = 15$. Since both T^b_2 and T^b_3 have the same minimum b-weight among $\{T^b_2, T^b_3, T^b_4\}$, we can choose T^b_2 or T^b_3 for T^b.

Until now, we have assumed that the minimum-b-weight spanning tree T^* contains a b-branch. It is possible, however, that T^* does not contain a b-branch at all. For this scenario, we can simply find a spanning tree $\widehat{T}_0$ with minimum weight $w(\widehat{T}_0)$ in $G_\phi \backslash \{\overline{i_0}\}$. Clearly, $\widehat{T}_0$ has minimum b-weight among all spanning trees that do not have a b-branch. In the case of our example, $\widehat{T}_0 = \{\bar{1},\bar{3},\bar{5},\bar{6},\bar{7}\}$ with a b-weight (weight) of 14.

Finally, after T^b and $\widehat{T}_0$ have been determined, we select for T^* the one with the lower b-weight. Since the entire search procedure is exhaustive in nature, T^* is obviously the optimal spanning tree for this problem.

Returning to our example, since $\widehat{T}_0$ has the lowest b-weight, we have $T^* = \widehat{T}_0 = \{\bar{1},\bar{3},\bar{5},\bar{6},\bar{7}\}$. Now equipped with T^*, an optimal tour τ^* can be obtained by $\tau^* = \phi \circ \langle 1,2\rangle \circ \langle 3,4\rangle \circ \langle 7,8\rangle \circ \langle 6,7\rangle \circ \langle 5,6\rangle = (1,3,7)(2,9)(4)(5,10)(6)(8) \circ \langle 1,2\rangle \circ \langle 3,4\rangle \circ \langle 7,8\rangle \circ \langle 6,7\rangle \circ \langle 5,6\rangle$. Note that we have followed the special order of the optimal pyramidal subtour $(5,8,7,6)$ while performing the transpositions of the branch $\mathcal{I}(5,7)$. As a result, an optimal solution for the TSP is given by the tour $1 \rightarrow 9 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 6 \rightarrow 10 \rightarrow 5 \rightarrow 8 \rightarrow 1$ with a length of $c(\phi) + w^b(T^*) = 803 + 14 = 817$.

The following algorithm is the straightforward summary of our overall solution strategy.

Algorithm b-Decomposable
Input: A b-decomposable permuted Monge matrix $C^{\phi} = (c_{i\phi(j)})$ with the permutation ϕ.
Output: An optimal TSP tour.
BEGIN
Construct the patching graph G_{ϕ};
Define the Monge matrix $D = (d_{ij})$ as $d_{ij} = c_{i\phi(j)}$;
Define the weights of the edges in G_{ϕ} by $w_i = d_{i,i+1} + d_{i+1,i} - d_{ii} - d_{i+1,i+1}$;
Sort the edges of G_{ϕ} into non-decreasing order by the weights w_i;
Find an MST $\widehat{T}$ of G_{ϕ} trying to delay the inclusion of edge $\overline{i_0}$ in the tree as long as possible;
IF $\widehat{T}$ does not contain edge $\overline{i_0}$ THEN
Patch all subtours of ϕ by using edge-transpositions from $\widehat{T}$ in the order of an optimal pyramidal subtour for each of its branches.
ELSE
Find a spanning tree $\widehat{T}_0$ with minimum weight in $G_{\phi} \setminus \{\overline{i_0}\}$;
Find a minimum-b-weight spanning tree T^b in G_{ϕ} among those containing a b-branch;
IF the weight of $\widehat{T}_0$ is less than the b-weight of T^b THEN
Patch all subtours of ϕ by using edge-transpositions from $\widehat{T}_0$ in the order of an optimal pyramidal subtour for each of its branches.
ELSE
Patch all subtours of ϕ by using edge-transpositions from T^b in the order of an optimal pyramidal subtour for each of its branches.
END

Now let us consider the running time of the algorithm. It takes $O(n \log n)$ time to sort the edges of G_{ϕ}. As previously described, $\widehat{T}$ and $\widehat{T}_0$ can be found by Kruskal's (1956) algorithm for the MST, which requires in this case only $O(n)$ time because the edges are in sorted order. Using the pointers, it takes $O(n^2)$ time to determine all T_h^b-s and T^b. The time to perform each of the remaining procedures is $O(n)$. Therefore, the running time for the entire algorithm is $O(n^2)$. Thus we have proved the following theorem.

Theorem 5.4 *Let $C = (c_{ij})$ be an $n \times n$ b-decomposable permuted Monge matrix with an optimal assignment ϕ. Then the TSP with distance matrix C is solvable in $O(n^2)$ time.*

5.3.2 A subclass with a faster solution

In the preceding section, we have studied TSP-s whose matrix can be decomposed into two b-weight-additive sub-matrices. In this section, we show that this TSP

is solvable in $O(n \log n)$ time if one of the two sub-matrices is a sum matrix. In particular, we establish that the bottleneck step of finding the minimum-b-weight spanning tree T^b in Algorithm b-Decomposable can be executed in linear time. Without loss of generality, we assume that the matrix $D' = (d'_{ij})$ for $i, j \leq i_0$ is the sum matrix.

First, it is interesting to note that if the distance matrix is a sum matrix, then for a given permutation ψ, the cost of performing any transposition on ψ is always zero. Thus, when D' is a sum matrix, the edges of the patching graph G_ϕ can be classified into two classes: The first class contains the edges $\bar{i}$ for $i \leq i_0 - 1$, and their weight is zero; the second class consists of the edges $\bar{i}$ for $i \geq i_0$, which have non-negative weight.

Now consider a series of edge-transpositions $\langle i, i+1\rangle, \langle i+1, i+2\rangle, \ldots, \langle j, j+1\rangle$ in G_ϕ with $i < i_0 - 1$ and $j \geq i_0$. If we first perform, in any order, all transpositions for $i \geq i_0 - 1$, the remaining transpositions can always be performed with zero cost. This observation implies that for any b-branch $\mathcal{I}(i, j)$ in G_ϕ with $i < i_0 - 1$, its b-weight is the same as that of $\mathcal{I}(i_0 - 1, j)$. Hence, while looking for T^b, it is unnecessary to consider those spanning trees that contain a b_h-branch with $h < i_0 - 1$. In other words, to find a minimum-b-weight spanning tree T^b_h in $G_\phi \backslash \{\overline{h-1}\}$ among all trees containing a b_h-branch, we need to consider only two cases, $h = i_0 - 1$ and i_0, if $i_0 > 1$ or only one case, $h = i_0$, if $i_0 = 1$. (In the example above, the two cases are $h = 3$ and 4.) Therefore, the tree T^b can be found in only $O(n)$ time and this yields a conceptually simpler solution for the problem studied by Aneja and Kamoun (1999).

Corollary 5.5 *Let $C = (c_{ij})$ be an $n \times n$ b-decomposable permuted Monge matrix in which one of the b-weight-additive components is a sum matrix. If an optimal assignment ϕ is given for C, then a minimum b-weight spanning tree and an optimal TSP tour can be found in $O(n \log n)$ time.*

Let us take a look again at the TSP studied by Aneja and Kamoun (1999) and the K-component robotic scheduling problem in Chapter 3. The Monge matrix $D = C^\phi = (c_{i\phi(j)})$ in these cases can be split into a sum sub-matrix and a Gilmore–Gomory sub-matrix. Since the optimal assignment ϕ can be found by sorting the b_i and a_i values so that $b_i \leq b_{i+1}$ and $a_{\phi(i)} \leq a_{\phi(i+1)}$, which requires $O(n \log n)$ time, it follows that these special cases of the TSP are solvable in $O(n \log n)$ time.

5.4 Summary

We studied the TSP on a special case of permuted Monge matrices, called b-decomposable matrices, where the corresponding Monge matrix can be partitioned into two b-weight-additive sub-matrices. The study of this case of the TSP was motivated by robotic scheduling problems. We have discussed how this new class of matrices can be recognized in polynomial time. Based on the subtour-patching

technique, we formulated the TSP on this special class of matrices as a minimum-b-weight spanning tree problem and described an $O(n^2)$ algorithm for it. Furthermore, we considered a special case of b-decomposable matrices whose one component is a sum sub-matrix, and showed that the optimal solution can be obtained faster for this case. As a byproduct of this, we have given a new algorithm and a simpler proof for the special TSP studied in Aneja and Kamoun (1999).

Chapter 6
Conclusions

"Science never solves a problem without creating ten more."
—George Bernard Shaw

We have studied several scheduling problems which generalize the classical robotic-cell scheduling models. Two aspects of our scheduling models were distinguished: reentrant processing and 1-K processing. We examined various robotic-cell problems with respect to these new types of processing. In this process, we have classified which problems in question are polynomially solvable and which are computationally intractable. Specifically, we showed that the problems are strongly $\mathcal{NP}$-hard with three machines and presented polynomial solutions for a variety of two-machine configurations. For the two-machine case, future developments may deal with more complex systems incorporating multiple robots, as well as taking processing time windows into account. *Processing time window* is a very general form of processing requirements that specify both the minimum and the maximum time a part can spend on a machine. Such constraints arise in a vast array of practical situations. For instance, in steel and chemical industry the duration of an annealing process can neither be too short nor be too long because it would affect the products' quality or characteristics.

A key factor in dealing with real-world robotic-cell problems is the development of accurate methods of modelling production in robotic cells. In practical applications, however, it is difficult to model all existing restrictions and costs properly in a mathematical model: It is known that the classical m-machine robotic-cell scheduling problems are already strongly $\mathcal{NP}$-hard when $m = 3$ (Hall, Kamoun, and Sriskandarajah 1998). The difficulty of the problems is compounded if additional constraints (e.g., release and due time) are involved. These constraints reflect requirements in certain real-life production environments. In this regard, a realistic approach is to decompose complex problems into relatively independent and tractable subproblems. Here we can perhaps take advantage of a hierarchical modelling strategy whereby a complicated problem is broken into pieces, each piece is modeled, and the pieces are stitched together in such a fashion that they form a comprehensive model for the underlying phenomenon—an issue which is worthy of further investigation. It is needed to point out that a number of the assumptions made in the analysis of reentrant and 1-K processing in the robotic cells, such as identical components and equal

number of components for all parts, may not always be valid in practice. However, these efforts have yielded considerable insights into the structure of these hitherto little examined problems, which might be used as a guideline for designing pragmatic solutions (e.g., heuristics) to larger problems. For example, the efficient algorithms for the two-machine case can offer a basis for fast practical algorithms when $m \geq 3$. Of course, extensive research on developing algorithms for the different types of part processing found in automated manufacturing environment, as well as computational experiments using real data, are required before solid conclusions as to the viability of these methods can be reached. Our work indicates this is a fruitful avenue to pursue.

There are numerous deterministic scheduling problems arising in the reentrant processing environment of which this thesis addressed only a few (Middendorf and Timkovsky 2002). We simply paid some attention to robotic-cells and problems which we believed to be particularly important and interesting. We hope that our work will spawn some interest along this line of research.

For the scheduling problem $J2|m_j \leq 3|C_t$, we have reduced it to the $RF2|\ell = 3|C_{\max}$ problem. This important result turns out to be very useful. By taking a closer look at the previous results on the $RF2|\ell = 3|C_{\max}$ problem, we provided extended results, which can be handily applied to $J2|m_j \leq 3|C_t$. Although the exact complexity status of the $RF2|\ell = 3|C_{\max}$ problem remains an open question, we now have a better understanding of the mysterious nature of the problem. Definitely, determining its complexity status will be of great importance because it sheds light on other hard open questions in job shop scheduling.

It is interesting to compare the two-machine job shop problem with cycle time and makespan objectives. When there is no idle time in the beginning of the schedule, the $J2|m_j \leq 3|C_t$ problem is easily seen to be identical to the makespan version of the problem, which is denoted by $J2|m_j \leq 3|C_{\max}$. The exact complexity status of the $J2|m_j \leq 3|C_{\max}$ problem is a long-standing open question (Lenstra, Rinnooy Kan, and Brucker 1977). In view of the above, one could suspect that the $J2|m_j \leq 3|C_{\max}$ problem can also be handled by a similar approach as for $J2|m_j \leq 3|C_t$. Recent results, however, suggest that one should be cautious here. Properties shared by an optimal schedule for the cycle time minimization problem may not carry over to the makespan minimization problem. It is worth mentioning that an optimal schedule for the $J2|m_j \leq 3|C_{\max}$ problem is not necessarily a permutation schedule—which perhaps makes this problem more intricate—in contrast to the situation for the $J2|m_j \leq 3|C_t$ problem. Here a job shop schedule is said to be a *permutation schedule* if in any stage the jobs with the same processing route are processed in the same sequence. See Drobouchevitch and Strusevich (1998) for details.

Jansen, Solis-Oba, and Sviridenko (2003) have recently studied the makespan minimization in an m-machine job shop where each job has at most k operations, i.e., the $Jm|m_j \leq k|C_{\max}$ problem. They described a linear time approximation scheme for it. It is not difficult to verify that this approximate result is applicable to the

cycle time version of the problem as well, i.e., the $Jm|m_j \leq k|C_t$ problem. (For an $\mathcal{NP}$-hard cost-minimization problem Π, an algorithm is said to be a *polynomial time approximation scheme* (PTAS) if for every instance I of Π and for every $\epsilon > 0$, it always returns a solution whose cost is at most $(1+\epsilon)$ times the optimal cost and its running time is bounded by a polynomial in the size of instance I.)

Finally, we have studied special cases of the traveling salesman problem on permuted Monge matrices, which arose from the robotic-cell scheduling problems. With respect to these special cases of the TSP, we defined a new class of matrices—b-weight-additive matrices—as a subclass of permuted Monge matrices for which the b-weight of any branch is equal to the total weight of the edges in that branch. Although these matrices can be recognized in $O(n^2)$ time combining Park's (1991) method for computing the b-weights with the algorithm for recognizing permuted Monge matrices, it remains an open question how to recognize and characterize them by algebraic properties. Another interesting question worthy of further probing is whether our polynomial-time solution approach can be extended to larger class of distance matrices.

Bibliography

Agnetis, A. 2000. Scheduling no-wait robotic cells with two and three machines. *European Journal of Operational Research* 123:303–314.

Agnetis, A., D. Pacciarelli, and F. Rossi. 1996. Lot scheduling in a two-machine cell with swapping devices. *IIE Transactions* 28:911–917.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network flows: Theory, algorithms, and applications.* Upper Saddle River, NJ: Prentice Hall.

Aneja, Y. P. and H. Kamoun. 1999. Scheduling of parts and robot activities in a two machine robotic cell. *Computers and Operations Research* 26:297–312.

Błażewicz, J., K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. 2001. *Scheduling computer and manufacturing processes.* 2d ed. Berlin, Germany: Springer-Verlag.

Brucker, P. 2001. *Scheduling algorithms.* 3d ed. Berlin, Germany: Springer-Verlag.

Burdyuk, V. Y. and V. N. Trofimov. 1976. Generalization of the results of Gilmore and Gomory on the solution of the traveling salesman problem. *Engineering Cybernetics* 14:12–18.

Burkard, R. E., V. G. Deĭneko, R. van Dal, J. A. A. van der Veen, and G. J. Woeginger. 1998. Well-solvable special cases of the traveling salesman problem: A survey. *SIAM Review* 40:496–546.

Burkard, R. E., B. Klinz, and R. Rudolf. 1996. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics* 70:95–161.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein. 2001. *Introduction to algorithms.* 2d ed. Cambridge, MA: The MIT Press.

Crama, Y., V. Kats, J. van de Klundert, and E. Levner. 2000. Cyclic scheduling in robotic flowshops. *Annals of Operations Research* 96:97–124.

Crama, Y. and J. van de Klundert. 1997. Cyclic scheduling of identical parts in a robotic cell. *Operations Research* 45:952–965.

Drobouchevitch, I. G. and V. A. Strusevich. 1998. Heuristics for short route job shop scheduling problems. *Mathematical Methods of Operations Research* 48:359–375.

Drobouchevitch, I. G. and V. A. Strusevich. 1999. A heuristic algorithm for two-machine re-entrant shop scheduling. *Annals of Operations Research* 86:417–439.

Elliott, D. J. 1989. *Integrated circuit fabrication technology.* 2d ed. New York, NY: McGraw-Hill.

Ernst, A. T., H. Jiang, M. Krishnamoorthy, and D. Sier. 2004. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153:3–27.

Ganesharajah, T., N. G. Hall, and C. Sriskandarajah. 1998. Design and operational issues in AGV-served manufacturing systems. *Annals of Operations Research* 76:109–154.

Garey, M. R. and D. S. Johnson. 1979. *Computers and intractability: A guide to the theory of NP-completeness.* San Francisco, CA: W. H. Freeman.

Gilmore, P. C. and R. E. Gomory. 1964. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research* 12:655–679.

Gilmore, P. C., E. L. Lawler, and D. B. Shmoys. 1985. Well-solved special cases. In *The traveling salesman problem: A guided tour of combinatorial optimization*, ed. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, 87–143. Chichester, England: John Wiley & Sons.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* 5:287–326.

Graves, S. C., H. C. Meal, D. Stefek, and A. H. Zeghmi. 1983. Scheduling of re-entrant flow shops. *Journal of Operations Management* 3:197–207.

Hall, N. G., H. Kamoun, and C. Sriskandarajah. 1997. Scheduling in robotic cells: Classification, two and three machine cells. *Operations Research* 45:421–439.

Hall, N. G., H. Kamoun, and C. Sriskandarajah. 1998. Scheduling in robotic cells: Complexity and steady state analysis. *European Journal of Operational Research* 109:43–65.

Hall, N. G., T.-E. Lee, and M. E. Posner. 2002. The complexity of cyclic shop scheduling problems. *Journal of Scheduling* 5:307–327.

Hertz, A., Y. Mottet, and Y. Rochat. 1996. On a scheduling problem in a robotized analytical system. *Discrete Applied Mathematics* 65:285–318.

Hurink, J. and S. Knust. 2001. Makespan minimization for flow-shop problems with transportation times and a single robot. *Discrete Applied Mathematics* 112:199–216.

Ioachim, I., E. Sanlaville, and M. Lefebvre. 2001. The basic cyclic scheduling model for robotic flow shops. *INFOR* 39:257–277.

Jansen, K., R. Solis-Oba, and M. Sviridenko. 2003. Makespan minimization in job shops: A linear time approximation scheme. *SIAM Journal on Discrete Mathematics* 16:288–300.

Johnson, S. M. 1954. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly* 1:61–68.

Kabadi, S. N. 2002. Polynomially solvable cases of the TSP. In *The traveling salesman problem and its variations*, ed. G. Gutin and A. P. Punnen, 489–583. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Kats, V. and E. Levner. 2002. Cyclic scheduling in a robotic production line. *Journal of Scheduling* 5:23–41.

Kise, H. 1991. On an automated two-machine flowshop scheduling problem with infinite buffer. *Journal of the Operations Research Society of Japan* 34:354–361.

Kise, H., T. Shioyama, and T. Ibaraki. 1991. Automated two-machine flowshop scheduling: A solvable case. *IIE Transactions* 23:10–16.

Kruskal, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* 7:48–50.

Kubiak, W., S. X. C. Lou, and Y. Wang. 1996. Mean flow time minimization in reentrant job shops with hub. *Operations Research* 44:764–776.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. 1993. Sequencing and scheduling: Algorithms and complexity. In *Logistics of production and inventory*, ed. S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, 445–522. Volume 4 of *Handbooks in operations research and management science.* Amsterdam, The Netherlands: Elsevier Science Publishers.

Lee, C.-Y., L. Lei, and M. Pinedo. 1997. Current trends in deterministic scheduling. *Annals of Operations Research* 70:1–41.

Lee, T.-E. and M. E. Posner. 1997. Performance measures and schedules in periodic job shops. *Operations Research* 45:72–91.

Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker. 1977. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1:343–362.

Lev, V. and I. Adiri. 1984. V-shop scheduling. *European Journal of Operational Research* 18:51–56.

Levner, E. and V. Kats. 1998. A parametric critical path problem and an application for cyclic scheduling. *Discrete Applied Mathematics* 87:149–158.

Levner, E., K. Kogan, and I. Levin. 1995. Scheduling a two-machine robotic cell: A solvable case. *Annals of Operations Research* 57:217–232.

Levner, E., K. Kogan, and O. Maimon. 1995. Flowshop scheduling of robotic cells with job-dependent transportation and set-up effects. *Journal of the Operational Research Society* 46:1447–1455.

Middendorf, M. and V. G. Timkovsky. 2002. On scheduling cycle shops: Classification, complexity and approximation. *Journal of Scheduling* 5:135–169.

Noble, P. J. W. 1989. *Printed circuit board assembly: The complete works.* New York, NY: Halsted Press.

Panwalkar, S. S. 1991. Scheduling of a two-machine flowshop with travel time between machines. *Journal of the Operational Research Society* 42:609–613.

Papadimitriou, C. H. 1994. *Computational complexity.* Reading, MA: Addison-Wesley Publishing Company.

Park, J. K. 1991. A special case of the n-vertex traveling-salesman problem that can be solved in $O(n)$ time. *Information Processing Letters* 40:247–254.

Pinedo, M. 2002. *Scheduling: Theory, algorithms, and systems.* 2d ed. Upper Saddle River, NJ: Prentice Hall.

Reddi, S. S. and C. V. Ramamoorthy. 1972. On the flow-shop sequencing problem with no wait in process. *Operational Research Quarterly* 23:323–331.

Sarvanov, V. I. 1980. On the complexity of minimizing a linear form on a set of cyclic permutations. *Soviet Mathematics - Doklady* 22:118–120.

Sethi, S. P., C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak. 1992. Sequencing of parts and robot moves in a robotic cell. *The International Journal of Flexible Manufacturing Systems* 4:331–358.

Sriskandarajah, C., N. G. Hall, and H. Kamoun. 1998. Scheduling large robotic cells without buffers. *Annals of Operations Research* 76:287–321.

Stankovic, J. A., M. Spuri, K. Ramamritham, and G. C. Buttazzo. 1998. *Deadline scheduling for real-time systems: EDF and related algorithms.* Boston, MA: Kluwer Academic Publishers.

van de Klundert, J. 1996. Scheduling problems in automated manufacturing. Ph.D. diss., University of Limburg, Maastricht, The Netherlands.

Vazirani, V. V. 2001. *Approximation algorithms.* Berlin, Germany: Springer-Verlag.

Wang, M. Y., S. P. Sethi, and S. L. van de Velde. 1997. Minimizing makespan in a class of reentrant shops. *Operations Research* 45:702–712.