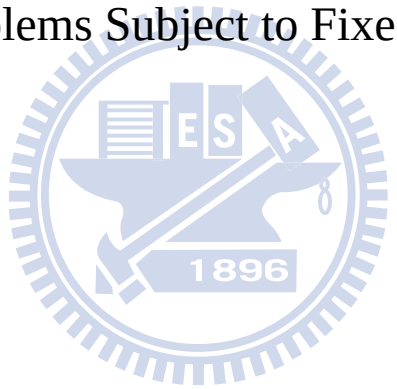
國立交通大學

資訊管理研究所

博士論文

固定工作順序條件下之排程問題

Scheduling Problems Subject to Fixed Job Sequences



研 究 生：黃鋒樟

指導教授：林妙聰 教授

中華民國 一百 年 一 月

固定工作順序條件下之排程問題

Scheduling Problems Subject to Fixed Job Sequences

研 究 生：黃鋒樟　　Student : Feng-Jang Hwang

指導教授：林妙聰　　Advisor : Bertrand Miao-Tsong Lin

國立交通大學

資訊管理研究所

博士論文

A Dissertation

Submitted to Institute of Information Management

College of Management

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Information Management

January 2011

Hsinchu, Taiwan, Republic of China

中華民國 一百 年 一 月

固定工作順序條件下之排程問題

研究生：黃鋒樟　　指導教授：林妙聰

國立交通大學資訊管理研究所博士班

摘　要

在生產排程問題中，工作或訂單順序代表其在機器上之處理先後順序，而排程則是明確排定每一工作或訂單在各台機器上之開始與完工時間。對某些排程問題而言，工作順序的給定並不直接等同排程結果。在這類型的問題中，除了排序之外，尚有諸如批量、交織穿插、延後執行等決策問題需要考量。這類型的排程問題即使固定其工作順序仍值得探究。在給定工作順序後，這類型的排程問題可稱之為固定工作順序條件排程問題。本論文針對三個固定工作順序條件排程問題進行研究。

本論文首先探究之固定工作順序條件排程議題為兩階段組裝線流線型機組批量排程問題。此問題考量的組裝線流線型機組環境為：階段一配置 m 台平行指定機器、階段二配置一台批量處理機器。給定 n 筆工作或訂單順序，並考量總完工時間最小化的目標下，本研究提出一個 $O(mn+n^5)$時間複雜度之演算法。

第二個研究主題為兩階段差異化流線型機組排程問題。此問題考慮的差異化流線型機組環境為：階段一配置一台公用機器、階段二配置 m 台平行指定機器。假設階段二的某平行指定機器 M_l 擁有 n_l 筆工作或訂單待處理。在分別給定 m 台平行指定機器個別之工作或訂單順序，並考量總完工時間最小化的目標下，本研究提出一個 $O(m^2\prod_{l=1}^{m} n_l^{m+1})$ 時間複雜度之動態規劃演算法。

第三個研究議題為單機器雙子作業排程問題。此問題的工作或訂單皆為雙子作業，每個工作的兩個子作業間存在一個固定的延遲時間。本研究之目標為固定工作順序條件下最大完工時間最小化。關於此議題下的固定工作順序條件排程問題，其時間複雜度仍懸而未決，然本研究提出一個 $O(n^2)$時間複雜度演算法解決固定“子作業”順序條件排程問題。此外，三個多項式時間可解的特例狀況亦在

本文中被提出探討。

關鍵字：固定工作順序排程問題；組裝線流線型機組；差異化流線型機組：雙子作業排程；動態規劃。

Scheduling Problems Subject to Fixed Job Sequences

Student：Feng-Jang Hwang　　Advisor：Bertrand Miao-Tsong Lin

Institute of Information Management
National Chiao Tung University

Abstract

In machine or shop scheduling, sequences of jobs or operations indicate the order of processing on machines while schedules explicitly specify the starting and completion times of activities on specific machines. For some problems, determining an optimal schedule from a given sequence may not be straightforward because another decision such as batching, interleaving, or idle time insertion is needed for optimality. In this study, three fixed-job-sequence problems are considered.

The first addressed problem is a two-stage assembly-type flowshop scheduling problem with batching considerations subject to a fixed job sequence. The assembly-type flowshop consists of m parallel dedicated machines at stage 1 and a batch machine at stage 2. The objective is to minimize the total completion time. A two-phase algorithm is developed to solve the studied problem in $O(mn+n^5)$ time, where n is the number of jobs and m is the number of parallel dedicated machines arranged at stage 1.

In the second problem, total completion time minimization in a two-stage differentiation flowshop subject to fixed sequences of jobs per type is studied. The two-stage differentiation flowshop comprises a stage-1 common machine and m stage-2 parallel dedicated machines. The goal is to determine an optimal interleaved processing sequence of all jobs at stage 1. This study presents an $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$ dynamic programming algorithm, where n_l is the number of type-l jobs. The running time is polynomial when m is constant.

In the third problem, the single-machine coupled-task scheduling, where the two tasks of each job are separated by an exact delay, is investigated. The aim is to schedule these coupled tasks to minimize the makespan subject to a given job sequence. Several intriguing properties of the studied problem are introduced. While the complexity status of the fixed-job-sequence problem remains open, an $O(n^2)$ algorithm is proposed to construct a feasible schedule attaining the minimum makespan for a given permutation of $2n$ tasks abiding by the fixed-job-sequence

constraint. Three polynomially solvable cases and a complexity graph of the fixed-job-sequence problem are presented.

Keywords: Fixed-job-sequence problem; Assembly-type flowshop; Differentiation flowshop; Coupled-task scheduling; Dynamic programming.

誌謝

古云：「弟子事師，敬同于父，習其道也，學其言語。」鋒樟首要感謝恩師 林教授妙聰，恩師在爲人處世與學術研究方面的啓迪與指導，惠余良多。恩師待人接物的圓融謙沖與治學態度的嚴謹踏實永遠是鋒樟的楷模與典範。四年的博士研究生涯，鋒樟無論在求學或生活上，皆受到恩師莫大的關照與協助，亦師亦友的情誼，愚徒永銘五內。

鋒樟感謝論文口試委員　葉維彰教授、李文炯教授、張永佳教授、劉敦仁教授、李永銘教授，在論文審查及口試期間辛勤地審查拙作，並惠予寶貴的建議與諸多的勉勵，愚學生無任感荷。

鋒樟感謝和藹的祖父　黃清圳先生與慈祥的先祖母　黃鍾美妹居士；親愛的父母親　黃明水先生與　黃夏秀琴女士，體諒鋒樟遊子求學在外，怠忽盡孝事親之責。同在清華大學生命科學院從事研究工作的兄長 黃國群博士，對愚弟無限的包容與照顧，鋒樟傾感不勝。

鋒樟尤甚虧欠與感激的是摯愛的妻女。賢內助 余國禎老師，無怨無悔地爲鋒樟分憂解勞，讓鋒樟能無後顧之憂浸心致力于研究，在鋒樟遭遇瓶頸困頓時給于鼓勵與支持，對鋒樟無限的包容與體諒，是鋒樟永遠的精神支柱。聰穎討喜的愛女Cathy是鋒樟的生命，是賢內助與鋒樟的小菩薩。

鋒樟感謝在交大一同奮鬥過的前輩與夥伴—和善樂天的故學長 健銘居士、熱心盡責的淑惠助理、善解人意的亞梅學姐、好施體貼的筱嵐學姊、相視莫逆的曜輝、樂觀積極的鴻順、溫柔貼心的怡菱、大方慷慨的彥廷、重情重義的癸堂、伶俐聰敏的欣穎、善良純眞的盈佑、資賦優異的鼎智、達觀開朗的彥徵、細心幽默的昕逸、多才多藝的慈徽、文靜乖巧的姿蓉、窮根究底

的秉陽、活力洋溢的建南、意氣風發的武澤、曠達率眞的以豪，因爲你們，方能有這段愉悅難忘的求學回憶。

楮墨有限，不盡欲言，需要感謝的人事物甚多，惟有感恩蒼天，期許劣者此生能爲杞梓之材。

黃 鋒 樟 謹誌

中華民國一百年元月于交通大學

Acknowledgments

I would like to express my sincere and deep gratitude to my advisor, Prof. Bertrand Miao-Tsong Lin, for his guidance, support, dedication, and encouragement in both academic and personal life through my Ph.D. study. Prof. Lin is such a gentleman that his enthusiasm and hard-working attitude to research makes his students want to present excellent performances. His energy, diligence and intelligence are inspiring.

I would also like to thank my Ph.D. dissertation committee – Prof. Yung-Chia Chang, Prof. Wen-Chiung Lee, Prof. Yung-Ming Li, Prof. Duen-Ren Liu and Prof. Wei-Chang Yeh, for their valuable comments and advices.

I would like to extend my grateful thanks to Prof. Sergey Sevastyanov, leading researcher at Sobolev Institute of Mathematics, Siberian Branch of the Russian Academy, for his significant advices in many discussions. In our research collaborations, I have learned a lot about the geometric methodology for scheduling problems. Especially, I want to show my gratitude to Prof. Mikhail Kovalyov, professor at United Institute of Informatics Problems, National Academy of Sciences of Belarus. The research collaborations with Prof. Kovalyov provides me a deep cognition of the concept of dynamic programming design. My gratefulness goes also to Prof. Alexander Kononov, senior researcher at Sobolev Institute of Mathematics, Siberian Branch of the Russian Academy. Prof. Kononov offered me an opportunity to explore my interest in approximation algorithms for scheduling problems.

Furthermore, I am grateful to Prof. Ming-Jong Yao, professor at Department of Transportation Technology and Management, National Chiao Tung University, for his advices and encouragement.

Finally, I would like to express my appreciation and gratitude to my beloved wife, Kuo-Jen Yu and my wonderful daughter, Catherine You-Han Hwang, for their tremendous love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Scheduling is a process of decision making about how to allocate the limited resources such as funds, raw materials, manpower, machines, and energy to the processing of projects or jobs for single- or multi-objective optimization. From the practical point of view, efficient scheduling plays a pivotal role in manufacturing and service industries owing to its indication of enterprise competitiveness. The application of scheduling can be found in production, transportation, communication, information processing, etc. In academic aspect, the theory of scheduling develops into a foundation of knowledge about mathematical models, combinatorial optimization, algorithmic methodologies, heuristic techniques and simulation approaches. Based on computational complexity theory, one significant branch in scheduling theory is devoted to investigating the efficient solvability of scheduling problems and the computing efforts required by their solution techniques (Baker & Trietsch, 2009). This research stream aims to explore whether the studied scheduling problems are intractable and determine their complexity statuses. The well-developed complexity hierarchies of scheduling problems can provide valuable information about the direction of effective problem-solving approaches and make a substantial contribution to industrial enterprises in real-world situations.

1.2 Motivation

Provided that a scheduling problem is proved to be intractable, i.e. NP-hard, it could be tackled by several approaches. One might adopt meta-heuristic strategies to achieve satisfactory near-optimal solutions in a reasonable time. Another might design approximation algorithms or a polynomial time approximation scheme (PTAS) to guarantee provable solution quality and run time bounds. A further approach is to investigate its various special cases and determine their complexity statuses. For some NP-hard scheduling problems, one of the special cases which are worthy of consideration could be the case with fixed sequence(s) of jobs.

In machine or shop scheduling, sequences of jobs or operations indicate the order of processing on machines while schedules explicitly specify the starting and completion times of activities on specific machines. In some cases, schedules can be directly determined by sequences of jobs or operations on the machines involved in the problems. In the other cases, extra information is needed to fully specify a schedule. The complicated schedule structure where a permutation/sequence of jobs/operations does not imply a schedule is paradigmatic of three major types of scheduling problems:

1. scheduling with batching;
2. scheduling with interleaving;
3. scheduling with inserted idle times.

In branch-and-bound, local search and meta-heuristic algorithms for tackling NP-hard scheduling problems of these types, development of efficient procedures for computing the incurred costs or objective values of complete or partial sequences of jobs is crucial to the efficiency of the solution approaches. In other words, within the local search procedure, the solution quality of candidate sequences of jobs needs to be assessed. Therefore, efficient algorithms for the fixed-job-sequence problems, if exist, can facilitate the development of meta-heuristics. Besides, the scheduling problems in which a specific optimal job sequence is established analytically can be exactly regarded as fixed-job-sequence problems. In real-life industrial applications, job sequence(s) can be included in input instances

since the fist-come-first-served (FCFS) principle is commonly regarded fair by customers. Additionally, a pre-assigned sequence of jobs could be retained on one of the machines in manufacturing process owing to technological or managerial decisions (Shafransky & Strusevich, 1998). Justification of the assumption of fixed job sequences can also be found in previous studies (Cheng et al., 2009, 2000b; Cheng & Wang, 1999; Herrmann & Lee, 1992; Hwang et al., 2010a; Kanet & Sridharan, 2000; Lin & Cheng, 2006, 2010; Lin et al., 2007; Ng & Kovalyov, 2007). The results conveyed in these previous works indicate that constructing optimal schedules from given job/operation sequences is not trivial for some scheduling problems.

1.3 State of the Art

Although several studies relative to "fixed-job-sequence scheduling" can be found in literatures, those results are not well emphasized, reviewed and categorized. The main reason that this research theme is not well explored seems to be a great diversity of these existing research results. Making a well-structured categorization for the research theme could be beneficial for its development. In the dissertation, three categories of fixed-job-sequence scheduling problems, including the problem of batching, the problem of interleaving, and the problem of idle time insertion, are proposed. The state of the art in the three categories of problems are briefly described as follows.

Batching decisions for a fixed job sequence in a flowshop were studied by Cheng et al. (2000b) and Ng & Kovalyov (2007) for makespan minimization, and by Hwang et al. (2010a) for total completion time minimization. Considering the fabrication and assembly of components in a two-machine flow shop, Cheng & Wang (1999) studied the makespan minimization problem of batching the jobs sequenced according to Johnson's rule or the agreeable processing time condition. Lin et al. (2007) investigated optimal batching of a fixed job sequence in a three-machine assembly-type flowshop for makespan minimization. Lin & Cheng (2010) also considered the objective function of maximum lateness, weighted number of tardy jobs or total weighted completion time for a fixed job sequence with centralized or decentralized batching decisions in concurrent open shops.

The problem commonly arising in the fixed-job-sequence scheduling where there exist several individual fixed job sequences or one job consists of several components is the interleaving issue. The first situation can be found in the differentiation flowshop environment. In a two-stage differentiation flowshop with a common machine at stage 1 and two parallel dedicated machines at stage 2, optimal interleaving of two fixed sequences per job type on the stage-1 machine was studied by Herrmann & Lee (1992) for the makespan minimization, and by Cheng et al. (2009) for the minimization of total weighted machine completion time. The second situation comes from the coupled-task scheduling (Shapiro, 1980), where the fixed-job-sequence problem could be worthwhile to investigate.

Scheduling with inserted idle times stems from just-in-time (JIT) production in which a non-regular performance measure such as the earliness-tardiness criterion is considered (Kanet & Sridharan, 2000). Algorithms for machine idle time insertion in a fixed job sequence, also called timing or timetabling algorithms (Hendel & Sourd, 2007), were applied for single machine (Bauman & Józefowska, 2006; Colina & Quinino, 2005; Davis & Kanet, 1993; Garey et al., 1988; Pan & Shi, 2005; Sourd, 2005; Szwarc & Mukhopadhyay, 1995), parallel machines (Della Croce & Trubian, 2002) and flowshop (Hendel & Sourd, 2007). In addition to scheduling with non-regular objective functions, the problem of idle time insertion also contains fixed-job-sequence scheduling with time-dependent processing times. Optimally timing a fixed job sequence in a two-machine flowshop in which the machine-2 processing time of each job depends on its waiting time between two machines was considered by Lin & Cheng (2006) for the makespan minimization, and by Hwang et al. (2010a) for the minimization of total completion time.

1.4 Research Issues

In this dissertation, three fixed-job-sequence problems are studied. The first addressed issue is a two-stage assembly-type flowshop scheduling problem with batching considerations subject to a fixed job sequence. There are m parallel dedicated machines arranged at stage 1, and stage 2 is equipped with a batch machine. The objective is to minimize the total completion time. In the second problem, total completion time minimization in a

two-stage differentiation flowshop subject to fixed sequences per job type is studied. The two-stage differentiation flowshop consists of a stage-1 common machine and m stage-2 parallel dedicated machines. The goal is to determine an optimal interleaved processing sequence of all jobs at stage 1. The third considered problem is the single-machine coupled-task scheduling where each job has two tasks separated by an exact delay. The objective is to schedule the tasks for makespan minimization subject to a given job sequence. The classification of the three studied problems is illustrated in Figure 1.1. With the common consideration of the fixed-job-sequence constraint, the linkage between the three distinctive studied problems is their affiliation to the same research theme.

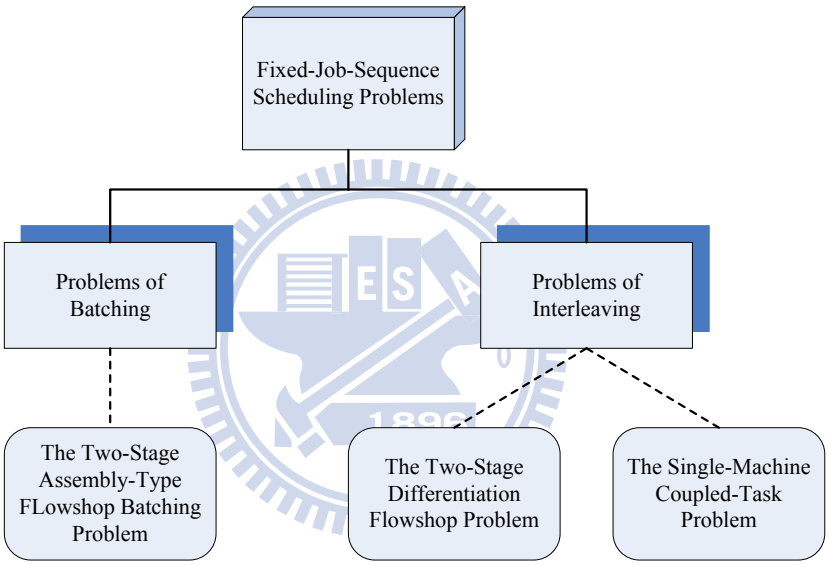


Figure 1.1: Categorization of the three studied problems.

1.5 Outline

This dissertation is organized into five chapters.

Chapter 2 describes the two-stage assembly-type flowshop batching problem subject to a fixed job sequence. After the general statements are given, a two-phase algorithm is described. Then a summary is provided.

In Chapter 3, the two-stage differentiation flowshop problem with fixed sequences per job type is introduced. The problem statements, the proposed dynamic programming

algorithm, and a summary are contained.

Chapter 4 considers the single-machine coupled-task problem with a fixed job sequence. Several intriguing properties of the problem are introduced. Then a polynomial time algorithm is presented to construct a schedule with the minimum makespan for a given *task* sequence abiding by the fixed-job-sequence constraint. Three polynomially solvable cases for the fixed-job-sequence problem are investigated and its complexity graph is given.

Concluding remarks are given in Chapter 5. This chapter presents some conclusions of this study. Finally, several recommendations for further research are also offered.

Chapter 2

Two-Stage Assembly-Type Flowshop Batch Scheduling

In this chapter, the minimization of total completion time in a two-stage assembly-type flowshop with batching considerations subject to a fixed job sequence is addressed. The goal is to obtain an optimal batching decision for the given job sequence at stage 2. This study presents a two-phase algorithm, which is developed by coupling a problem-transformation procedure with a dynamic program. The running time of the proposed algorithm is $O(mn + n^5)$, where m is the number of parallel dedicated machines arranged at stage 1 and n is the number of jobs.

The main contribution of the research is to demonstrate the polynomial solvability of the performance measure of total completion time for the studied problem. Base upon the proposed dynamic programming concept, the potential conflicts between the makespan and the total completion time of the subschedule can be avoided. Our results are achieved under the consideration of arbitrary m-machine case, and the proposed methodology can be exploited for any specific machine configuration in the studied scheduling model.

2.1 Preliminaries

The scheduling model considered in this study is a two-stage assembly-type flowshop, which is a generalization of Johnson's two-machine flowshop. The typical example of

the assembly-type flowshop scheduling is automobile assembly, such as the fire engine production (Lee et al., 1993). A fire engine comprises three major component parts, i.e. the body, the chassis and the engine. These three parts are produced by three parallel dedicated machines and then delivered to an assembly line for final assembly operations, as illustrated in Figure 2.1. Consider a set of n jobs or orders to be processed in a two-stage flowshop equipped with $m + 1$ machines. At stage 1, there are m parallel dedicated machines which independently produce component parts for jobs. Then these component parts are transferred to the stage-2 assembly line for assembly operation. Each job consists of $m + 1$ specific operations to be executed respectively on the m stage-1 parallel dedicated machines and the stage-2 assembly machine. At stage 2, the assembly operations are processed in batches, and the batch availability sum-batch (or sequential-batch) model with a non-anticipatory constant setup time is assumed. The batch availability indicates that jobs of the same batch complete at the same time, when processing of the latest job in this batch has been finished. In the sum-batch model, the processing time of a batch is defined as the sum of the setup time and the processing times of all jobs belonging to this batch. The non-anticipatory setup implies that a setup can start only after all the component parts of the jobs in the same batch are transferred to stage 2 and the stage-2 assembly machine is not occupied. We also assumed that the centralized decision making policy is adopted. Namely, all the m parallel dedicated machines comply with the sequencing and batching decisions determined by the assembly organization. Accordingly, they begin their first operation processing simultaneously and process the jobs consecutively without inserted idle times. Under the centralized decision making policy, the fixed job sequence considered in this study is predetermined by the assembly organization and followed by the m parallel dedicated machines. The objective is to minimize the total completion time. Following Lin et al. (2007), this study denotes the considered problem by $(m+1)\text{MAF}|m\delta \rightarrow \beta, \textit{sum-batch}, \textit{fixed\_seq}|\sum C_j$, where $(m + 1)$MAF stands for $(m + 1)$-machine assembly flowshop, $m\delta \rightarrow \beta$ for a two-stage system with m stage-1 discrete processors and a stage-2 batch processor, *sum-batch* for sum-batch model, and $\textit{fixed\_seq}$ for fixed job sequence. Even for the base case $m = 2$, the

3MAF$|2\delta \rightarrow \beta, sum\text{-}batch| \sum C_j$ problem is strongly NP-hard since it is a generalization of the strongly NP-hard problem $F2|\delta \rightarrow \beta, sum\text{-}batch| \sum C_j$.

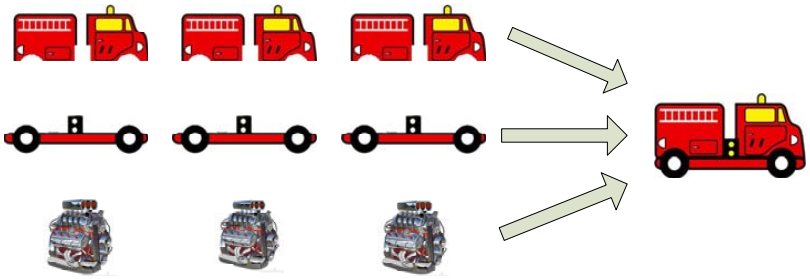


Figure 2.1: Assembly operation for fire engine production.

Motivation for the studied problem comes from the supply chain management in which the coordination of production scheduling between cooperative companies can be modelled as an assembly-type production scheduling problem (Lin et al., 2007). The parallel dedicated machines at stage 1 can be regarded as individual upstream suppliers which produce component parts or materials for the downstream manufacturer. The downstream company works as the stage-2 assembly organization which performs its production in batches. The studied scheduling model can be applied to the multi-product packing problem which commonly exists in the snack-food industry (Portougal, 1997). Many snack-food manufacturing companies offer variety packs of several different flavors of snack products and the content and size of the variety pack can be custom-made. Then the two-stage assembly flowshop for the variety pack of m different flavors consists of m production lines for different flavors of snack products at stage 1 and a batching machine for packing at stage 2. Another similar application can be found in a multi-page invoice printing system (Zhang et al., 2010). The production process between the page printing stage and the invoice assembly stage in the simplified invoice production system can be regarded as the assembly-type flowshop scheduling. For a three-page invoice, jobs are processed in three parallel dedicated printing lines at stage 1 and then conveyed to the stage-2 assembly line for invoice assembly.

Various models involving batching considerations have been proposed and investigated in the scheduling literature. Allahverdi et al. (1999, 2008); Cheng et al. (2000a); Potts & Kovalyov (2000) provided comprehensive surveys and reviews on this subject. Lee et al.

(1992) proposed a so-called burn-in model for semiconductor manufacturing, describing the processing time of a batch as the longest processing time of the jobs in the batch. This batching with simultaneous job processing is known as max-batch (parallel-batch) type. Following the sum-batch model, Cheng et al. (2000b) investigated a two-machine flowshop batching model with both batch processors for makespan minimization. They presented a strong NP-hardness proof, polynomial algorithms for several special cases, and some heuristics. A similar model was considered by Glass et al. (2001), who assume that the batch setup on machine 2 is anticipatory and that the setup time is machine-dependent. The strong NP-hardness proof and a heuristic with a worst-case performance ratio of 4/3 were provided. Cheng & Wang (1998) studied the makespan minimization problem in a two-machine flowshop comprising a discrete processor and a batch processor. In their model, the jobs are processed individually on machine 1, and processed in batches on machine 2. The authors proved ordinary NP-completeness of the problem and presented algorithms for some polynomially solvable cases. Subsequently, a strong NP-hardness result of the problem was proved by Lin & Cheng (2005). Problem $(m+1)\text{MAF}|m\delta \rightarrow \beta|C_{\max}$ was previously studied by Kovalyov et al. (2004), who investigated both the max-batch and sum-batch scenarios. A heuristic algorithm and a performance ratio analysis were presented. Lin et al. (2007) studied the $3\text{MAF}|2\delta \rightarrow \beta, \textit{sum-batch}|C_{\max}$ problem and proved its strong NP-hardness.

Potts & Kovalyov (2000) indicated that dynamic programming is utile for the single-machine batching problems where the sequencing and batching decisions can be decoupled. However, for shop models the fixed job sequence is necessary to design dynamic programs. Considering a given job sequence in a two-machine flowshop with both batch processors, Cheng et al. (2000b) developed an $O(n^3)$ algorithm to minimize the makespan. An $O(n^5)$ algorithm was proposed by Hwang et al. (2010a) for the same machine setting with the minimization of total completion time. For the makespan minimization, an $O(n^{5m-7})$ dynamic programming algorithm for the generalized m-machine environment was presented by Ng & Kovalyov (2007). For the fabrication and assembly scheduling in a two-machine flow shop, each job consists of three components: a common component and a unique

component which are both executed on machine 1, and an assembly component which is executed on machine 2 after the above two components are completed. Common components of all jobs are executed in batches, each of which is preceded by the same setup time. For the makespan minimization in the identical common component case, Cheng & Wang (1999) proposed an $O(n^4)$ algorithm to optimally batch the jobs sequenced according to Johnson's rule. For the constant assembly time case, another $O(n^3)$ algorithm was developed for optimally batching the jobs sequenced according to the agreeable processing time condition. For the performance measure of total completion time, Hwang et al. (2010b) designed an $O(n^7)$ dynamic program for the general case. As for the assembly flowshop batching problem, Lin et al. (2007) proposed an $O(n^2)$ dynamic program for problem $3\text{MAF}|2\delta \rightarrow \beta, \textit{sum-batch}, \textit{fixed\_seq}|C_{\max}$. A brief summary of complexity results of related fixed-job-sequence flowshop batching problems is given in Table 2.1.

Table 2.1: Complexity results of related fixed-job-sequence flowshop batching problems.

Problem	Complexity	Reference
$F2\|\textit{sum-batch}, \textit{fixed\_seq}\|C_{\max}$	$O(n^3)$	Cheng et al. (2000b)
$Fm\|\textit{sum-batch}, \textit{fixed\_seq}\|C_{\max}$	$O(n^{5m-7})$	Ng & Kovalyov (2007)
$F2\|\textit{sum-batch}, \textit{fixed\_seq}\|\sum C_j$	$O(n^5)$	Hwang et al. (2010a)
$F2\|\textit{sum-batch}, (c, u_j, a_j), \textit{fixed\_seq}\|C_{\max}$	$O(n^4)$	Cheng & Wang (1999)
$F2\|\textit{sum-batch}, (c_j, u_j, a), \textit{fixed\_seq}\|C_{\max}$	$O(n^3)$	Cheng & Wang (1999)
$F2\|\textit{sum-batch}, (c_j, u_j, a_j), \textit{fixed\_seq}\|\sum C_j$	$O(n^7)$	Hwang et al. (2010b)
$3\text{MAF}\|2\delta \rightarrow \beta, \textit{sum-batch}, \textit{fixed\_seq}\|C_{\max}$	$O(n^2)$	Lin et al. (2007)

2.2 Problem Definition

In this section, the formal problem definition is provided for the base case $m = 2$. A demonstration that the case with arbitrary m parallel dedicated machines can be easily generalized by the formulation of the base case will be given in the next section.

Assume without loss of generality that a given sequence of jobs $(1, 2, \ldots, n)$ is to be processed in a two-stage assembly flowshop with two dedicated parallel machines, M_a and M_b, at stage 1 and one assembly machine M_2 at stage 2. Each job j consists of three operations to be processed on M_a, M_b and M_2, respectively. The corresponding processing times are $p_{a,j}$, $p_{b,j}$ and $p_{2,j}$, respectively. Denote $p_{a,[i:j]} = \sum_{h=i}^{j} p_{a,h}$, $p_{b,[i:j]} = \sum_{h=i}^{j} p_{b,h}$,

and $p_{2,[i:j]} = \sum_{h=i}^{j} p_{2,h}$. After both stage-1 operations of job j are completed, these two produced component parts of job j are transferred to stage 2 for assembly. M_2 processes the jobs in batches with a non-anticipatory constant setup time s. The k-th batch formed in stage 2 is denoted by B_k. The objective is to optimally batch the given job sequence for the minimization of total completion time. Consider the following instance for illustration: $(p_{a,1}, p_{b,1}, p_{2,1}) = (2, 1, 1)$, $(p_{a,2}, p_{b,2}, p_{2,2}) = (1, 3, 3)$, $(p_{a,3}, p_{b,3}, p_{2,3}) = (4, 2, 2)$, $(p_{a,4}, p_{b,4}, p_{2,4}) = (1, 3, 1)$, $(p_{a,5}, p_{b,5}, p_{2,5}) = (2, 2, 2)$, $(p_{a,6}, p_{b,6}, p_{2,6}) = (5, 3, 3)$, and $s = 1$. Assume that the batching decision is to group jobs $\{1, 2\}$ into B_1, jobs $\{3, 4, 5\}$ into B_2, and $\{6\}$ into B_3. The obtained schedule is illustrated in Figure 2.2 and $\sum C_j = 90$.

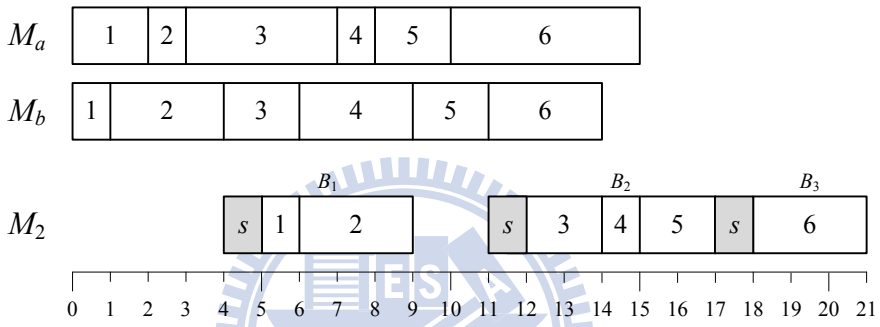


Figure 2.2: Example schedule.

2.3 Two-Phase Algorithm

This section introduces a two-phase algorithm developed for the $(m+1)\text{MAF}|m\delta \rightarrow \beta, \textit{sum-batch}, \textit{fixed\_seq}|\sum C_j$ problem. The first phase is a preprocessing procedure utilized to transform the studied problem to a two-machine flowshop batching problem subject to a fixed job sequence. Notice that given the instance data, the schedule of jobs at stage 1 can be settled in the studied problem. Then problem $(m+1)\text{MAF}|m\delta \rightarrow \beta, \textit{sum-batch}, \textit{fixed\_seq}|\sum C_j$ can be transformed to problem $F2|\delta \rightarrow \beta, \textit{sum-batch}, \textit{fixed\_seq}|\sum C_j$ by mapping the stage-1 schedule of the original problem to the stage-1 discrete processor in the transformed problem. Consider the case with $m = 2$. As illustrated in Figure 2.3, the schedule of a *dummy* discrete processor M_1 can be mapped from that of the two parallel dedicated machines M_a and M_b by setting the

completion time of job j on M_1 as $C_{1,j} = \max\{p_{a,[1:j]}, p_{b,[1:j]}\}$. Then the studied problem is transformed to a two-machine flowshop problem with a discrete processor M_1 at stage 1 and a batch machine M_2 at stage 2. If m is constant, then $O(n)$ time is needed in the first phase. For the general case of arbitrary m, the first phase requires $O(mn)$ time.

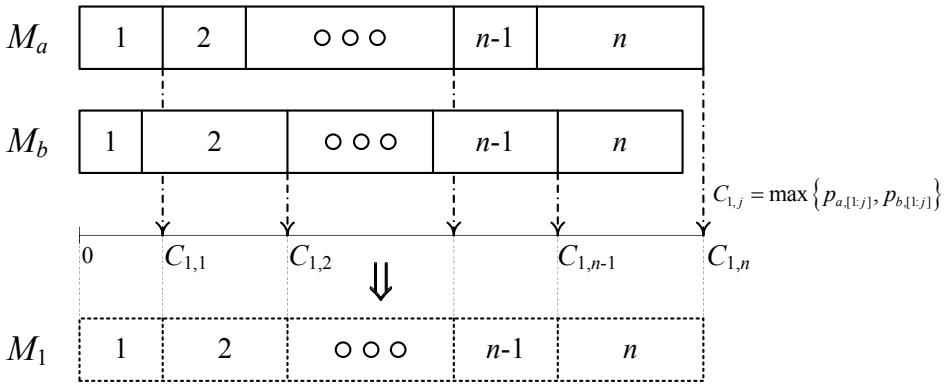


Figure 2.3: Mapping of the schedule of M_a and M_b to that of M_1.

In the second phase, problem $F2|\delta \rightarrow \beta, sum\text{-}batch, fixed\_seq|\sum C_j$ is coped with by a dynamic program. The difficulty in the design of a polynomial time dynamic program arises from the potential conflicts between the makespan and the total completion time. A subschedule of the first j jobs that minimizes the total completion time may have a comparatively large makespan, which will worsen the total completion time of the remaining $n - j$ jobs. To resolve the problem, a dynamic program incorporating one state variable to specify possible makespans can be developed. Nevertheless, the time complexity of the dynamic program designed by this approach will be pseudo-polynomial time. The technique devised in this study is to introduce a fixed number of jobs or positional indices to specify makespans.

For a partial schedule, a maximal (by inclusion) sequence of stage-2 batches processed consecutively without inserted idle times is denoted by a *block*. The last block of a considered partial schedule is called *critical block*. A partial schedule of jobs $1, 2, \ldots, j$ is defined by a state (i, i_1, j, k), where

1) i and j are respectively the first and the last jobs of the critical block,

2) k is the number of batches in the critical block, and

3) i_1 is the last job of the first batch in the critical block.

The structure of a partial schedule (i, i_1, j, k) is shown in Fig. 2.4. Notice that several partial schedules can be associated with the same state. Let $f(i, i_1, j, k)$ denote the minimum total completion time in a partial schedule among those associated with the same state (i, i_1, j, k) for $1 \le i \le i_1 \le j \le n$ and $\lceil \frac{j-i_1}{j} \rceil + 1 \le k \le j - i_1 + 1$. The development of the proposed dynamic program is based upon forward recursions by batches, i.e. the last batch of the partial schedule is removed for each recursion. The dynamic programming formulation is given in a pseudocode-like fashion.

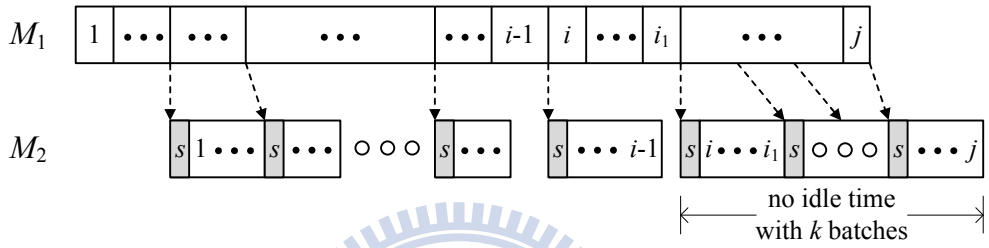


Figure 2.4: Illustration of a partial schedule (i, i_1, j, k).

Algorithm DP-Batch

Initialization:

`For` each feasible i, i_1, j, k satisfying $1 \le i \le i_1 \le j \le n$, $\lceil \frac{j-i_1}{j} \rceil + 1 \le k \le j - i_1 + 1$,

$$f(i, i_1, j, k) = \begin{cases} j(C_{1,j} + s + p_{2,[1:j]}), & \text{if } i = k = 1 \text{ and } i_1 = j; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursion:

$f(i, i_1, j, k)$ is determined by two disjoint cases: $k = 1$ (Figure 2.5) and $k > 1$ (Figure 2.6).

`For` each feasible i, i_1, j, k satisfying $2 \le i \le i_1 \le j \le n$, $\lceil \frac{j-i_1}{j} \rceil + 1 \le k \le j - i_1 + 1$ do

Case $k = 1$ (also implying $i_1 = j$):

Set $z = \infty$;

`For` each feasible i', i'_1, k' satisfying $1 \le i' \le i'_1 \le i - 1$, $\lceil \frac{i-i'_1-1}{i-1} \rceil + 1 \le k' \le i - i'_1$ do

`If` $C_{1,j} - C_{1,i'_1} > k's + p_{2,[i':i-1]}$

`then` $temp = f(i', i'_1, i-1, k') + (j - i + 1)(C_{1,j} + s + p_{2,[i:j]})$;

`else` $temp = \infty$;

$z = \min\{z, temp\}$.

$f(i, i_1, j, k) = z$.

Case $k > 1$:

Set $z = \infty$;

`For` each j' from $i_1 + k - 2$ up to $j - 1$ do

`If` $C_{1,j} - C_{1,i_1} \leq (k-1)s + p_{2,[i:j']}$

`then` $temp = f(i, i_1, j', k-1) + (j - j')(C_{1,i_1} + ks + p_{2,[i:j]})$;

`else` $temp = \infty$;

$z = \min\{z, temp\}$.

$f(i, i_1, j, k) = z$.

Goal: Find $\min\{f(i, i_1, n, k) \mid 1 \leq i \leq i_1 \leq n,\ \lceil \frac{n-i_1}{n} \rceil + 1 \leq k \leq n - i_1 + 1\}$.

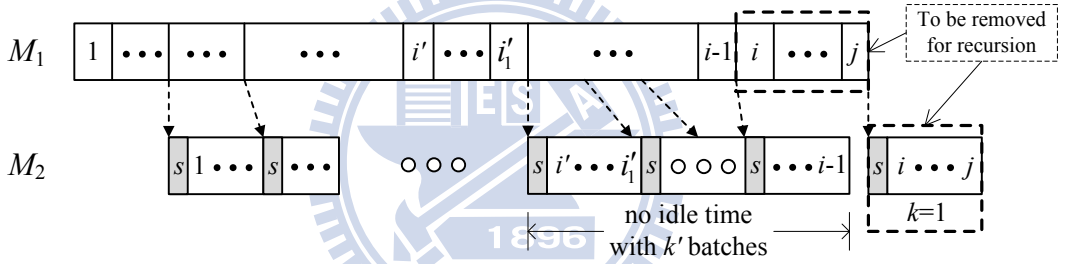


Figure 2.5: Forward recursion of the case $k = 1$.

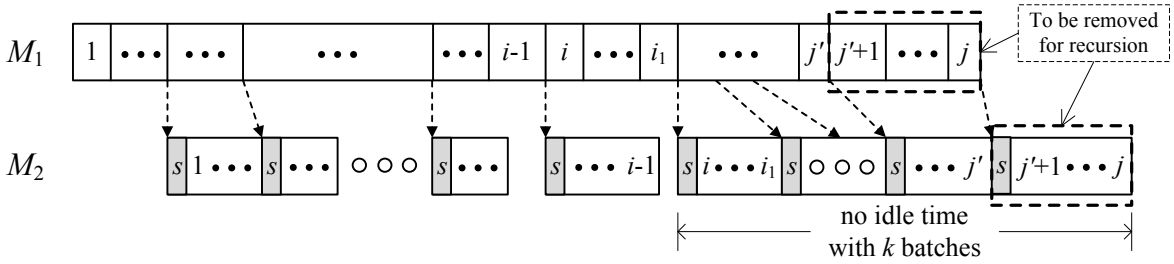


Figure 2.6: Forward recursion of the case $k > 1$.

Algorithm DP-Batch consists of two cases: 1) For the case $k = 1$, there are $O(n^2)$ states $(i, j, j, 1)$, each of which requires $O(n^3)$ operations; 2) For the case $k > 1$, there are $O(n^4)$ states (i, i_1, j, k), each of which needs $O(n)$ operations. The Goal step requires $O(n^3)$ comparisons, each of which takes constant time. Hence, the running time of algorithm

DP-Batch is $O(n^5)$. Since the first and second phases respectively require $O(mn)$ and $O(n^5)$ times, the running time of the proposed two-phase algorithm is $O(mn + n^5)$.

Theorem 2.1. *Problem* $(m+1)MAF|m\delta \rightarrow \beta, sum\text{-}batch, fixed\_seq|\sum C_j$ *can be solved in* $O(mn + n^5)$ *time.* □

Example. Consider the following instance with $m = 2$ and $n = 4$: $(p_{a,1}, p_{b,1}, p_{2,1}) = (2, 1, 1)$, $(p_{a,2}, p_{b,2}, p_{2,2}) = (1, 3, 3)$, $(p_{a,3}, p_{b,3}, p_{2,3}) = (4, 2, 2)$, $(p_{a,4}, p_{b,4}, p_{2,4}) = (1, 3, 1)$, and $s = 1$. The two-phase algorithm is demonstrated as follows:

Phase 1. (Problem-transformation procedure)

$C_{1,1} = \max\{2, 1\} = 2;$

$C_{1,2} = \max\{2 + 1, 1 + 3\} = 4;$

$C_{1,3} = \max\{2 + 1 + 4, 1 + 3 + 2\} = 7;$

$C_{1,4} = \max\{2 + 1 + 4 + 1, 1 + 3 + 2 + 3\} = 9.$

Phase 2. (Algorithm DP-Batch)

Initialization:

$f(1, 1, 1, 1) = 1 \times (2 + 1 + 1) = 4;$

$f(1, 2, 2, 1) = 2 \times (4 + 1 + 4) = 18;$

$f(1, 3, 3, 1) = 3 \times (7 + 1 + 6) = 42;$

$f(1, 4, 4, 1) = 4 \times (9 + 1 + 7) = 68;$

For other values of i, i_1, j, k, we denote $f(i, i_1, j, k) = \infty$.

Recursion:

$f(2, 2, 2, 1) = \infty;$

$f(2, 2, 3, 2) = f(2, 2, 2, 1) + (3 - 2)(C_{1,2} + 1 \times 1 + p_{2,[2:3]}) = \infty;$

$f(2, 2, 4, 2) = \min\{\infty, f(2, 2, 3, 1) + (4 - 3)(C_{1,2} + 2 \times 1 + p_{2,[2:4]})\} = \infty;$

$f(2, 2, 4, 3) = f(2, 2, 3, 3) + (4 - 3)(C_{1,2} + 3 \times 1 + p_{2,[2:4]}) = \infty;$

$f(2, 3, 3, 1) = f(1, 1, 1, 1) + (3 - 2 + 1)(7 + 1 + 5) = 30;$

$f(2, 3, 4, 2) = f(2, 3, 3, 1) + (4 - 3)(7 + 2 + 6) = 30 + 15 = 45;$

$f(2, 4, 4, 1) = f(1, 1, 1, 1) + (4 - 2 + 1)(9 + 1 + 6) = 4 + 48 = 52;$

$f(3, 3, 3, 1) = \infty;$

$f(3,3,4,2) = f(3,3,3,1) + (4-3)(C_{1,3} + 2 \times 1 + p_{2,[3:4]}) = \infty;$

$f(3,4,4,1) = \min\{f(1,1,2,2) + (4-3+1)(9+1+3), \infty, f(2,2,2,1) + (4-3+1)(9+1+3)\} = \infty;$

$f(4,4,4,1) = \infty.$

Goal:

$\min\{f(i,i_1,4,k) \mid 1 \le i \le i_1 \le 4, \lceil \frac{4-i_1}{4} \rceil + 1 \le k \le 4 - i_1 + 1\}$

$= \min\{f(2,3,4,2), f(2,4,4,1)\} = 45.$

The optimal schedule $(2,3,4,2)$ can be constructed by backtracking the recursion, as demonstrated in Figure 2.7.

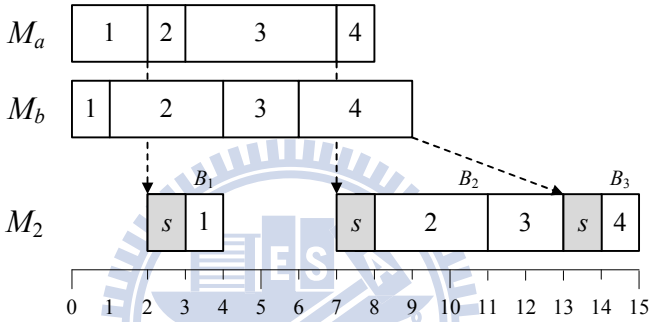


Figure 2.7: Optimal schedule $(2,3,4,2)$.

2.4 Summary

A two-stage assembly-type flowshop batching problem with a fixed job sequence has been addressed in this study. For the minimization of total completion time, this study designed an $O(mn + n^5)$-time two-phase algorithm, where m is the number of parallel dedicated machines arranged at stage 1 and n is the number of jobs. The running time will be $O(n^5)$ if the number of dedicated machines m is not a part of the input. Besides, problem $F2|\delta \rightarrow \beta, \textit{sum-batch}, \textit{fixed\_seq}|\sum C_j$ was also solved in $O(n^5)$ time in this study. Furthermore, the developed algorithm can be easily generalized for the weighted counterparts.

Further study could be conducted on the max-batch model, i.e. $(m+1)\text{MAF}|m\delta \rightarrow \beta, \textit{max-batch}, \textit{fixed\_seq}|\sum C_j$. Other performance measures, such as the maximum lateness and the total tardiness could also be considered.

Chapter 3

Two-Stage Differentiation Flowshop Scheduling

This chapter addresses total completion time minimization in a two-stage differentiation flowshop where the sequences of jobs per type are predetermined. The two-stage differentiation flowshop consists of a stage-1 common machine and m stage-2 parallel dedicated machines. The goal is to determine an optimal interleaved processing sequence of all jobs at the first stage. This study presents an $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$ dynamic programming algorithm, where n_l is the number of type-l jobs. The running time is polynomial when m is constant.

The main research contribution is to investigate the performance merit of total completion time, which had not been considered in literatures for the studied problem. The uniqueness of the proposed dynamic programming algorithm is the design of matrix state variables. With the consideration of arbitrary m-machine case, the presented methodology can be utilized for any specific machine configuration in the studied problem.

3.1 Problem Statements

This chapter considers a two-stage differentiation flowshop scheduling problem to minimize the total completion time, subject to the condition that job sequences per type are known a priori. The two-stage differentiation flowshop consists of a stage-1 common ma-

chine M_0 and m stage-2 parallel dedicated machines, $M_1, \ldots, M_m$. Jobs are categorized into m types, and the number of type-l jobs is n_l for $1 \leq l \leq m$. All jobs are required to be processed on machine M_0 first, and then jobs of type l proceed to dedicated machine M_l for their second-stage process. Since the processing sequence of each job type is given, the goal is to find an interleaved processing sequence of all jobs on machine M_0 so as to minimize the sum of job completion times at stage 2 of all jobs. The problem under study is denoted by $F(1,m)|fixed\_seq|\sum C_j$, where $F(1,m)$ stands for a two-stage differentiation flow shop with m parallel dedicated machines at stage 2, $fixed\_seq$ for fixed sequences of jobs per type, and $\sum C_j$ for the total completion time minimization criterion. To solve the $F(1,m)|fixed\_seq|\sum C_j$ problem optimally, this study presents a dynamic programming algorithm with a running time that is polynomial when the number of dedicated machines m is constant.

The studied problem is motivated by a production-and-painting system (Mosheiov & Sarig, 2010). All products are manufactured by the stage-1 common machine. Then each fabricated product is delivered to its dedicated machine for the specific color painting at stage 2. A practical application is furniture manufacturing, such as chairs (Cheng et al., 2009). At stage 1, the main body of the chair is produced by a common production line. At stage 2, m types of head-supports or armrests are assembled on the main bodies by m parallel dedicated machines. Another example is the pottery shaping (Cheng et al., 2009), as illustrated in Figure 3.1. The main glazing process of potteries is performed on a common machine at stage 1. To possess distinct appearances or figures, the potteries need to be processed with different heating treatments, e.g. low-temperature firing and high-temperature firing. Potteries with various features are made with different firing processes on the corresponding dedicated machines at stage 2.

Herrmann & Lee (1992) first studied the $F(1,2)$ model (two job types) and showed the strong NP-hardness of three objectives, namely the makespan, the number of tardy jobs and the total completion time. An interesting problem arising from the machine configuration is to determine an optimal interleaved sequence on the stage-1 machine from fixed sequences for the two types of jobs. This interleaving problem of makespan minimization

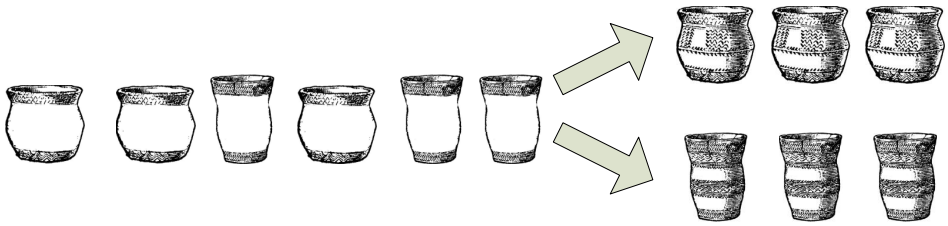


Figure 3.1: Configuration of pottery production.

was reduced to the problem of minimizing the maximum lateness, which can be solved by Jackson's earliest due date (EDD) first rule (Jackson, 1955) in $O((n_1 + n_2)\log(n_1 + n_2))$ time. Kyparisis & Koulamas (2000) and Mosheiov & Yovel (2004) proposed polynomial time algorithms for the $F(1, m)$ problem of makespan minimization subject to the block assumption that jobs of the same type must be processed adjacently on the stage-1 machine. With the block assumption, Mosheiov & Sarig (2010) investigated the $F(1, m)$ model to minimize the weighted number of tardy jobs with a common due date and proposed a pseudo-polynomial dynamic programming algorithm to establish the ordinary NP-hardness. Cheng & Kovalyov (1998) considered the $F(1, 2)$ model incorporating batching decisions on the common machine, where setup times occur whenever the machine switches processing from a job of one type to a job of the other type. A polynomial time dynamic programming algorithm for makespan minimization was presented. Cheng et al. (2009) addressed a non-classical objective of minimizing the weighted sum of stage-2 machine completion times, which is denoted by $F(1, 2)||\text{WMT}$. They proved the strong NP-hardness and designed an $O(n^3)$ time algorithm for the special case with given sequences of both types of jobs. The solution approach developed by Herrmann & Lee (1992) for problem $F(1, 2)|fixed\_seq|C_{\max}$ actually can be further extended for the general problem $F(1, m)|fixed\_seq|C_{\max}$. A brief summary of complexity results of the fixed-job-sequence differentiation flowshop problems is given in Table 3.1.

Table 3.1: Complexity results of fixed-job-sequence differentiation flowshop problems.

Problem	Complexity	Reference
$F(1, 2)\|fixed\_seq\|C_{\max}$	$O((n_1 + n_2)\log(n_1 + n_2))$	Herrmann & Lee (1992)
$F(1, m)\|fixed\_seq\|C_{\max}$	$O(\sum_{l=1}^{m} n_l \log \sum_{l=1}^{m} n_l)$	§
$F(1, 2)\|fixed\_seq\|\text{WMT}$	$O(n^3)$	Cheng et al. (2009)

§ The result is derived from Herrmann & Lee (1992).

To the best of our knowledge, the objective of total completion time minimization was not previously addressed in the differentiation flowshop problems. Due to the equivalence of $F(1,1)||\sum C_j$ and the strong NP-hard $F2||\sum C_j$ problem, the $F(1,m)||\sum C_j$ problem is also intractable. Investigating problem $F(1,m)|fixed\_seq|\sum C_j$ is thus appropriate. The preliminary study suggests that even the $F(1,2)|fixed\_seq|\sum C_j$ problem cannot be solved using the approach developed for problem $F(1,2)|fixed\_seq|C_{\max}$ (Herrmann & Lee, 1992).

3.2 The Proposed Dynamic Program

Denote by $\mathcal{J}_l = \{J_{l,1}, \ldots, J_{l,n_l}\}$ the set of type-l jobs, $1 \leq l \leq m$. Job $J_{l,j}$ requires a processing time $p_{l,j}$ and $q_{l,j}$ on machine M_0 and M_l, respectively. The processing sequence of jobs per type is already predetermined. Assume without loss of generality that the fixed sequence of type-l jobs is $(J_{l,1}, J_{l,2}, \ldots, J_{l,n_l})$.

Let us first consider a special case where each type contains exactly one job, i.e. $n_l = 1$ for all types l. In this case, we denote p_l and q_l the processing times of the type-l job. Consider a particular sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$. The completion time of the j-th job is given by $\sum_{i=1}^{j} p_{\sigma_i} + q_{\sigma_j}$. The total completion time is thus given by

$$\sum_{j=1}^{m}\left(\sum_{i=1}^{j} p_{\sigma_i} + q_{\sigma_j}\right) = \sum_{j=1}^{m}\sum_{i=1}^{j} p_{\sigma_i} + \sum_{j=1}^{m} q_{\sigma_j}.$$

The second term is fixed once the instance is given. Therefore, the problem is equivalent to minimizing $\sum_{j=1}^{m}\sum_{i=1}^{j} p_{\sigma_i}$, which can be solved in $O(m \log m)$ time by the shortest processing time (SPT) first rule using p_l.

For the general case, this study proposes a dynamic programming algorithm in which two matrices, A and B are designed. Define matrix $\mathsf{A}_{1\times m} = [a_1, a_2, \ldots, a_m]$ with $0 \leq a_l \leq n_l$ for $1 \leq l \leq m$, where the element a_l is the number of the type-l job(s) in the considered subschedule. Given A, the objective is to find the optimal interleaved sequence of subsequences $(J_{1,1}, \ldots, J_{1,a_1}), \ldots, (J_{m,1}, \ldots, J_{m,a_m})$. In a given subschedule of

this problem, the last job having an idle time on its machine at stage 2 inserted in prior to its dedicated operation is called a *critical job* of its type. Matrix $\mathsf{B}_{m\times m}$ is defined with $0 \le b_{l,r} \le a_r$ for $1 \le l, r \le m$, where the element $b_{l,r}$ is defined as the number of the type-r job(s) arranged on machine M_0 before the stage-1 completion time of the critical job of type l. A subschedule is associated with a state $(k, \mathsf{A}, \mathsf{B})$ subject to the following conditions: (a) Subsequences $(J_{1,1}, \ldots, J_{1,a_1}), \ldots, (J_{m,1}, \ldots, J_{m,a_m})$ are considered; (b) Job J_{k,a_k} is the last job on machine M_0 and $a_k \neq 0$; (c) Job $J_{l,b_{l,l}}$ is the critical job of type $l, 1 \le l \le m$, and its completion on machine M_0 is preceded by jobs of $\bigcup_{r=1}^{m}\{J_{r,1}, \ldots, J_{r,b_{l,r}}\}$; (d) If $a_l \neq 0$, $1 \le b_{l,l} \le a_l$; otherwise $b_{l,r} = 0$. As depicted in Figure 3.2, the configuration is aimed at determining the stage-2 completion time of the job scheduled last on M_0. With the parameter specifications, the completion time of job J_{k,a_k} is calculated as $C_{k,a_k} = \sum_{r=1}^{m}\sum_{j=1;b_{k,r}\neq 0}^{b_{k,r}} p_{r,j} + \sum_{j=b_{k,k}}^{a_k} q_{k,j}$. Consider an instance with $m = 3$: $(p_{1,1}, p_{1,2}, p_{1,3}) = (2, 5, 4)$, $(q_{1,1}, q_{1,2}, q_{1,3}) = (8, 4, 7)$, $(p_{2,1}, p_{2,2}, p_{2,3}, p_{2,4}) = (4, 5, 5, 3)$, $(q_{2,1}, q_{2,2}, q_{2,3}, q_{2,4}) = (8, 4, 17, 3)$, $(p_{3,1}, p_{3,2}, p_{3,3}) = (3, 4, 6)$, $(q_{3,1}, q_{3,2}, q_{3,3}) = (8, 7, 4)$. A schedule of the state with $k = 2$, $\mathsf{A} = [3, 4, 3]$, and $\mathsf{B} = \begin{bmatrix} 3 & 2 & 0 \\ 3 & 3 & 0 \\ 3 & 3 & 1 \end{bmatrix}$ is shown in Figure 3.3, and we have $C_{2,4} = \sum_{r=1}^{3}\sum_{j=1;b_{2,r}\neq 0}^{b_{2,r}} p_{r,j} + \sum_{j=b_{2,2}}^{a_2} q_{2,j} = 25 + 20 = 45$.

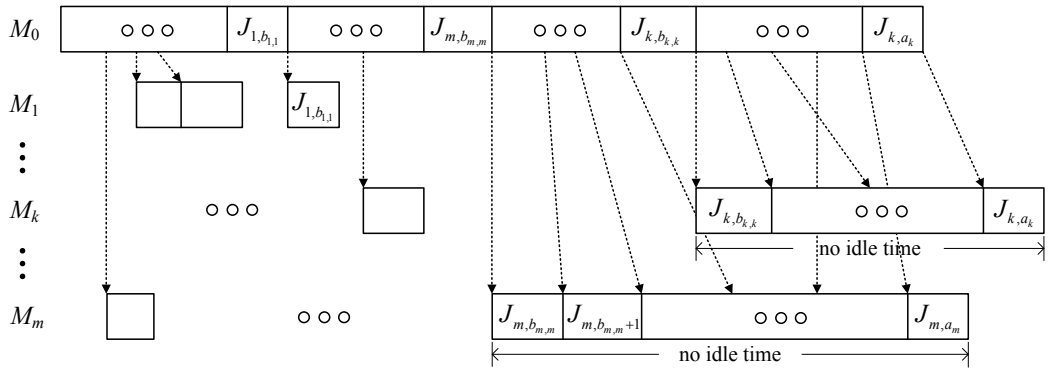


Figure 3.2: Configuration of state $(k, \mathsf{A}, \mathsf{B})$.

The corresponding recursive function $f_k(\mathsf{A}, \mathsf{B}), 1 \le k \le m$ is defined as the minimum total completion time among all schedules associated with the same state $(k, \mathsf{A}, \mathsf{B})$. From the above definition, the recursive formulation of the proposed dynamic program is given

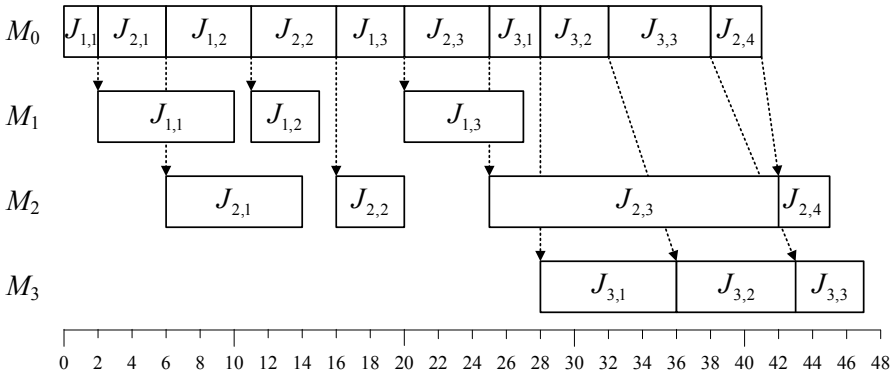


Figure 3.3: Instance schedule of the state $(2, \mathsf{A}, \mathsf{B})$.

as follows. Note that given a matrix A, the range of possible vales of $b_{l,r}$ for $1 \leq l, r \leq m$ can be obtained.

Algorithm DP-F1m

Initialization: For all $k \in \{1, \ldots, m\}$, all matrices A, and all matrices B,

$$f_k(\mathsf{A}, \mathsf{B}) = \begin{cases} 0, & \text{if } \mathsf{A} \text{ and } \mathsf{B} \text{ both are zero matrices;} \\ \infty, & \text{otherwise.} \end{cases}$$

Recursion: For $1 \leq k \leq m$, each matrix A satisfying $1 \leq a_k \leq n_k$, and $0 \leq a_l \leq n_l$ for $1 \leq l \neq k \leq m$, and each possible matrix B corresponding to A, perform the recursion by removing the last job J_{k,a_k}.

Define the updated matrix A' by letting $a'_k = a_k - 1$, and $a'_l = a_l$ for $1 \leq l \neq k \leq m$.

Case 1 ($b_{k,k} = a_k$):

In this case, J_{k,a_k} is the critical job of its type. After each recursive call, the critical job of type k needs to be updated. Construct the updated matrix B' by letting $b'_{l,r} = b_{l,r}$, $1 \leq l \neq k \leq m$ and $b'_{k,r}$ denote the number of the type-r job(s) arranged on machine M_0 before the stage-1 completion time of the updated critical job of type k for $1 \leq r \leq m$.

Subcase 1-1 ($b_{k,k} = a_k > 1$):

Denote the set $\mathcal{D} = \{[b'_{k,1}, b'_{k,2}, \ldots, b'_{k,m}] : 1 \leq b'_{k,k} < b_{k,k}$, and $0 \leq b'_{k,r} \leq b_{k,r}$ for $1 \leq r \neq k \leq m\}$. In this subcase, we have the k-th row of B', $[b'_{k,1}, b'_{k,2}, \ldots, b'_{k,m}] \in \mathcal{D}$. The

configuration of the subcase is presented in Figure 3.4.

$$f_k(\mathsf{A},\mathsf{B}) = \min_{1\le l\le m;\mathcal{D}} \left\{ f_l(\mathsf{A'},\mathsf{B'}) : \sum_{r=1}^{m} \sum_{j=b'_{k,r}+1; b'_{k,r}\neq a'_r}^{a_r} p_{r,j} > \sum_{j=b'_{k,k}}^{a'_k} q_{k,j} \right\} + C_{k,a_k}.$$

Subcase 1-2 ($b_{k,k} = a_k = 1$):

In this subcase, J_{k,a_k} is the unique job of its type as well. After removing J_{k,a_k} for recursion, there exists no critical job of type k, i.e. $a'_k = 0$. By virtue of the aforementioned condition (d), we have $b'_{k,r} = 0$ for $1 \le r \le m$. The configuration of the subcase is presented in Figure 3.5.

$$f_k(\mathsf{A},\mathsf{B}) = \min_{1\le l\le m} \left\{ f_l(\mathsf{A'},\mathsf{B'}) \right\} + C_{k,a_k}.$$

Case 2 ($b_{k,k} < a_k$):

For this case, J_{k,a_k} is not the critical job of its type. As illustrated in Figure 3.6, we simply remove J_{k,a_k} in the recursion.

$$f_k(\mathsf{A},\mathsf{B}) = \min_{1\le l\le m} \left\{ f_l(\mathsf{A'},\mathsf{B}) : \sum_{r=1}^{m} \sum_{j=b_{k,r}+1; b_{k,r}\neq a'_r}^{a_r} p_{r,j} \le \sum_{j=b_{k,k}}^{a'_k} q_{k,j} \right\} + C_{k,a_k}.$$

Goal: Let $a_l = n_l$ for $1 \le l \le m$. Find $\min_{1\le k\le m} \left\{ f_k(\mathsf{A},\mathsf{B}) : 0 \le b_{l,r} \le n_r \text{ and } 1 \le b_{l,l} \le n_l \text{ for } 1 \le l, r \le m, r \ne l \right\}$.

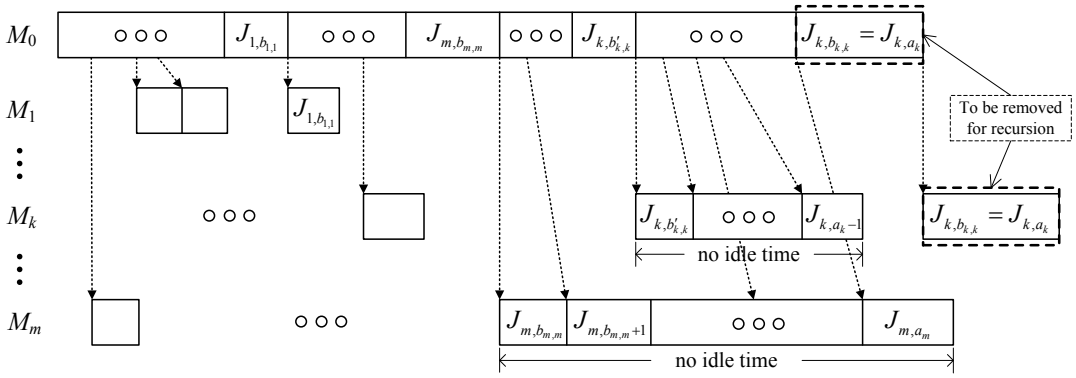


Figure 3.4: Recursion of the subcase $b_{k,k} = a_k > 1$.

As for the complexity of algorithm DP-F1m, the running times for Subcase 1-1, Subcase 1-2, Case 2, and the Goal phase are analyzed as follows. For Case 1, $b_{k,k} = a_k$ implies

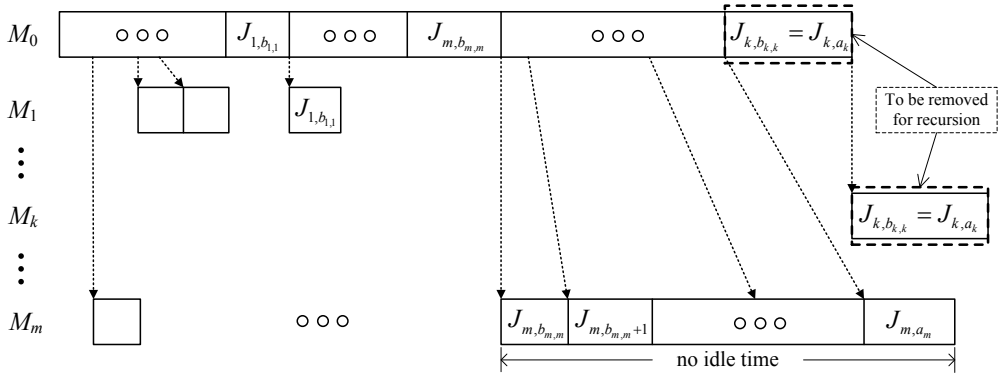


Figure 3.5: Recursion of the subcase $b_{k,k} = a_k = 1$.

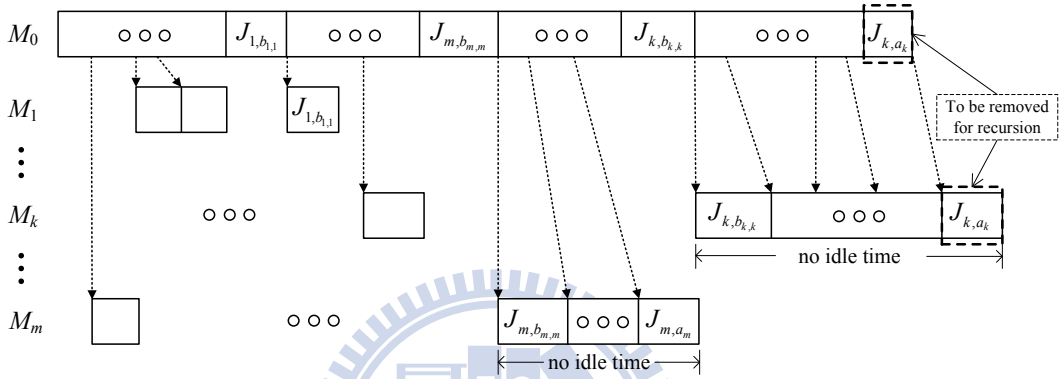


Figure 3.6: Recursion of the case $b_{k,k} < a_k$.

that $b_{k,r} = a_r$, $1 \le r \le m$. Hence, there are $O(m \prod_{l=1}^{m} n_l^m)$ states, each of which takes $O(m \prod_{l=1}^{m} n_l)$ time in Subcase 1-1. The running time is $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$. In Subcase 1-2, $a_k = 1$ implies that $b_{l,k} = 0$, $1 \le l \ne k \le m$, and there are less than $O(m \prod_{l=1}^{m} n_l^m)$ states, each of which takes $O(m)$ time. The running time is thus $O(m^2 \prod_{l=1}^{m} n_l^m)$. In Case 2, the size of the state space is $O(m \prod_{l=1}^{m} n_l^{m+1})$ and the computation required for each state takes $O(m)$ time. It results in a total running time of $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$ for the Recursion phase. When the Recursion phase is done, the Goal phase requires $O(m \prod_{l=1}^{m} n_l^m)$ comparisons, each of which takes constant time. Therefore, the overall running time of algorithm DP-F1m is $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$, which is polynomial when m is not a part of the input. For the specific case of $m = 2$, the complexity is $O(n_1^3 n_2^3)$.

Theorem 3.1. *Problem* $F(1,m)|fixed\_seq|\sum C_j$ *can be solved in* $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$ *time, where* n_l *is the number of type-l jobs.* □

Example. Consider the following instance with $m = 2$, $n_1 = 2$ and $n_2 = 2$: $(p_{1,1}, p_{1,2}) = (2,5)$, $(q_{1,1}, q_{1,2}) = (1,4)$, $(p_{2,1}, p_{2,2}) = (4,3)$, and $(q_{2,1}, q_{2,2}) = (3,2)$. Algorithm DP-F1m

is demonstrated as follows:

Initialization:

$f_1([0,0], \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}) = f_2([0,0], \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}) = 0;$

For other values of $k, \mathsf{A}, \mathsf{B}$, we denote $f_k(\mathsf{A}, \mathsf{B}) = \infty$.

Recursion:

$(1) k = 1, 1 \le a_1 \le 2, a_2 = 0$:

$f_1([1,0], \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}) = 3;$

$f_1([2,0], \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}) = \infty;$

$f_1([2,0], \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}) = f_1([1,0], \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}) + 11 = 14.$

$(2) k = 2, a_1 = 0, 1 \le a_2 \le 2$:

$f_2([0,1], \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}) = 7;$

$f_2([0,2], \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}) = f_2([0,1], \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}) + 9 = 16;$

$f_2([0,2], \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}) = \infty.$

$(3) k = 1, a_1 = 1, a_2 = 1$:

$f_1([1,1], \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}) = f_2([0,1], \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}) + 7 = 14;$

$(4) k = 2, a_1 = 1, a_2 = 1$:

$f_2([1,1], \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}) = f_1([1,0], \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}) + 9 = 12;$

$(5) k = 1, a_1 = 2, a_2 = 1$:

$f_1([2,1], \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}) = \infty;$

$f_1([2,1], \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}) = \infty.$

$f_1([2,1], \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}) = f_1([1,1], \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}) + 15 = 29;$

$f_1([2,1], \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}) = f_2([1,1], \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}) + 15 = 27.$

$(6) k = 1, a_1 = 1, a_2 = 2$:

$$f_1([1,2],\begin{bmatrix}1 & 2\\ 0 & 1\end{bmatrix}) = f_2([0,2],\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}) + 10 = 26;$$

$$f_1([1,2],\begin{bmatrix}1 & 2\\ 0 & 2\end{bmatrix}) = \infty.$$

(7)$k = 2, a_1 = 1, a_2 = 2$:

$$f_2([1,2],\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}) = f_2([1,1],\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}) + 11 = 23;$$

$$f_2([1,2],\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}) = \infty.$$

$$f_2([1,2],\begin{bmatrix}1 & 0\\ 1 & 2\end{bmatrix}) = \infty;$$

$$f_2([1,2],\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}) = f_1([1,1],\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}) + 11 = 25.$$

(8)$k = 2, a_1 = 2, a_2 = 1$:

$$f_2([2,1],\begin{bmatrix}1 & 0\\ 2 & 1\end{bmatrix}) = \infty;$$

$$f_2([2,1],\begin{bmatrix}2 & 0\\ 2 & 1\end{bmatrix}) = f_1([2,0],\begin{bmatrix}2 & 0\\ 0 & 0\end{bmatrix}) + 14 = 28.$$

(9)$k = 1, a_1 = 2, a_2 = 2$:

$$f_1([2,2],\begin{bmatrix}1 & 2\\ 0 & 1\end{bmatrix}) = \infty;$$

$$f_1([2,2],\begin{bmatrix}1 & 2\\ 0 & 2\end{bmatrix}) = \infty;$$

$$f_1([2,2],\begin{bmatrix}2 & 2\\ 0 & 1\end{bmatrix}) = f_1([1,2],\begin{bmatrix}1 & 2\\ 0 & 1\end{bmatrix}) + 18 = 44;$$

$$f_1([2,2],\begin{bmatrix}2 & 2\\ 0 & 2\end{bmatrix}) = \infty;$$

$$f_1([2,2],\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}) = \infty;$$

$$f_1([2,2],\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}) = \infty;$$

$$f_1([2,2],\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}) = \infty;$$

$$f_1([2,2],\begin{bmatrix}1 & 0\\ 1 & 2\end{bmatrix}) = \infty;$$

$$f_1([2,2],\begin{bmatrix}2 & 2\\ 1 & 1\end{bmatrix}) = f_2([1,2],\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}) + 18 = 41;$$

$$f_1([2,2],\begin{bmatrix}2 & 2\\ 1 & 2\end{bmatrix}) = f_2([1,2],\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}) + 18 = 43;$$

$(10) k = 2, a_1 = 2, a_2 = 2$:

$$f_2([2,2],\begin{bmatrix}1 & 0\\ 2 & 1\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}2 & 0\\ 2 & 1\end{bmatrix}) = f_2([2,1],\begin{bmatrix}2 & 0\\ 2 & 1\end{bmatrix}) + 16 = 44;$$

$$f_2([2,2],\begin{bmatrix}1 & 0\\ 2 & 2\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}2 & 0\\ 2 & 2\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}2 & 1\\ 0 & 1\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}1 & 1\\ 2 & 2\end{bmatrix}) = \infty;$$

$$f_2([2,2],\begin{bmatrix}2 & 1\\ 2 & 2\end{bmatrix}) = f_1([2,1],\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}) + 16 = 43;$$

Goal:

$$\min_{1\le k\le 2}\left\{f_k([2,2],\mathsf{B}) : 0 \le b_{l,r} \le 2 \text{ and } 1 \le b_{l,l} \le 2 \text{ for } 1 \le l, r \le 2, r \ne l\right\} = f_1([2,2],\begin{bmatrix}2 & 2\\ 1 & 1\end{bmatrix}) = 41.$$

The optimal schedule $(1,[2,2],\begin{bmatrix}2 & 2\\ 1 & 1\end{bmatrix})$ can be constructed by backtracking the recursion, as demonstrated in Figure 3.7.

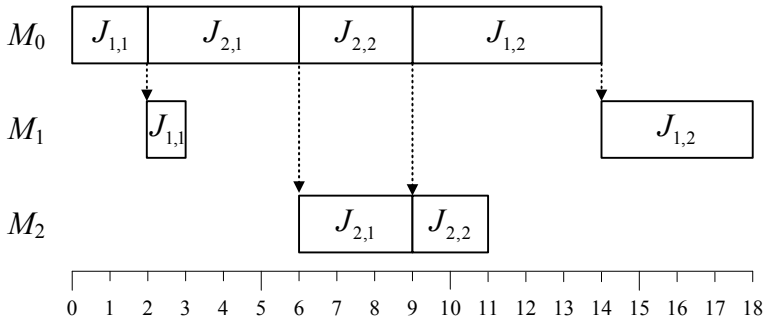


Figure 3.7: Optimal schedule for the example.

3.3 Summary

A two-stage differentiation flowshop scheduling problem with predetermined job sequences per type has been addressed in this study. For the minimization of the total completion time, we designed an $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$-time dynamic programming algorithm, where n_l is the number of type-l jobs. The running time is polynomial when the number of dedicated machines m is constant.

Two directions are suggested for further extensions of our research. First, since the stage-1 machine is common for all product types, in the aspect of mass customization the processing is mostly carried out in batches. It would be interesting to consider different batching modes, including max-batch (parallel-batch) and sum-batch (sequential-batch) on the common machine. Second, we can consider the reverse model that has two dedicated machines at stage 1 and a common machine at stage 2. The differentiation flowshop and its reverse model are equivalent for makespan minimization, but exhibit different characteristics for the total completion time minimization.

Chapter 4

Single-Machine Coupled-Task Scheduling

In this chapter, single-machine coupled-task scheduling where each job has two tasks separated by an exact delay is investigated. The objective of this study is to schedule the tasks to minimize the makespan subject to a given job sequence. Several intriguing properties of the fixed-job-sequence problem under study are introduced. While the complexity status of the studied problem remains open, an $O(n^2)$ algorithm is proposed to construct a feasible schedule attaining the minimum makespan for a given permutation of $2n$ tasks abiding by the fixed-job-sequence constraint. Several polynomially solvable cases of the fixed-job-sequence problem are investigated and a complexity graph of the problem is presented.

The contribution of the study is the consideration of the fixed-job-sequence assumption, which could be suitable in practical applications, for the studied problem. Although the considered problem is not solved, a polynomial time algorithm is presented for its subproblem in which a task sequence abiding by the fixed-job-sequence constraint is predetermined. Furthermore, three polynomially solvable cases for the studied problem are demonstrated.

4.1 General Statements

This study considers the problem of scheduling coupled tasks with exact delays, i.e. each job consists of two distinct tasks which are separated by a fixed time interval. Coupled-task scheduling, also known as the two-phased job scheduling problem (Sherali & Smith, 2005), primarily stems from operations scheduling of pulsed radar systems (Orman et al., 1998). A pulsed radar system is utilized to detect and locate objects. The typical example is that the airplanes approaching a congested airport are tracked for the terminal-area air traffic management. A radar transmits a pulse which will reflect back after reaching an object, and then receives the echo signal after a specified time period. To track an aircraft, a radar system repeats the transmission and reception operations. The pulse transmission and reception cannot occur at the same time. Neither can any two transmission or reception operations overlap. For multi-target tracking, the pulse transmission and reception operations, which are regarded as coupled tasks, shall be scheduled to minimize the idle time for the radar system. Another application is the command-and-control system in the parallel computing environment (Sherali & Smith, 2005). To solve a large-scale computational problem, a command unit divides the problem into several subproblems and distributes them to several client units for numerical calculations. Each client unit send back its calculation result to the command unit after the computation. The aim is to obtain the solution of the original problem which is a combination of the solutions of those subproblems in the minimum computational time.

Assume a set of n two-phased jobs $\{J_1, J_2, \ldots, J_n\}$ to be processed on a single machine. Each two-phased job J_j consists of two separate tasks that require processing times a_j and b_j, respectively. If no confusion would arise, a_j and b_j are also used to denote the two tasks of job J_j. Under the constraint of exact delays, the starting time of the second task b_j of any job J_j *must* be exactly l_j time units after the completion of its first task a_j. The problem, denoted as $1|\text{Coup-Task}|C_{\max}$ by Orman & Potts (1997) and $1|\text{exact } l_j|C_{\max}$ by Ageev & Kononov (2006), is to find a feasible schedule such that the makespan is minimized. This problem is known to be strongly NP-hard even in some special cases (Orman & Potts, 1997). This study aims to investigate coupled-

task scheduling subject to a given job sequence. Herewith, given a fixed job sequence, if job J_i precedes job J_j in the specified sequence, then it is required to schedule the tasks such that a_i precedes a_j, and b_i precedes b_j. We denote the studied problem by $1|(a_j, l_j, b_j), fjs|C_{\max}$, where "$fjs$" in the second field dictates the assumption of a fixed job sequence.

The first study on coupled-task scheduling with exact delays could be due to Shapiro (1980), who established that problem $1|(a_j, l_j, b_j)|C_{\max}$ is equivalent to the NP-hard job-shop problem $J2|\textit{no-wait}, M_2\ \textit{non-bott}|C_{\max}$, where "*no-wait*" and "M_2 *non-bott*" respectively refer to the no-wait constraint and the infinite processing capacity of the second machine. Three polynomial-bounded heuristics for numerical experiments were also presented. Orman & Potts (1997) investigated the complexity of several special cases of problem $1|(a_j, l_j, b_j)|C_{\max}$. All the analyzed cases are classified to be strongly NP-hard or polynomially solvable, except for the case with identical coupled tasks, $1|(a, l, b)|C_{\max}$. Ahr et al. (2004) proposed a dynamic programming algorithm based on a directed graph model for this special case with time complexity $O(nr^{2l})$, where $r \leq \sqrt[a-1]{a}$. The algorithm is linear in the number of jobs only for fixed l and is not polynomial in the input size which is measured by $\log a + \log l + \log b + \log n$. Then Baptiste (2010) showed that the case can be solved in $O(\log n)$ when a, l, b are fixed. To the best of our knowledge, the complexity status of identical coupled-task scheduling problem remains open. Blazewicz et al. (2010) studied problem $1|(1, l, 1), prec|C_{\max}$ with strict precedence constraints and proved its NP-hardness in the strong sense. They also proposed an $O(n)$ algorithm for the special case where $l = 2$ and an in-tree or out-tree precedence constraints graph are assumed. Ageev & Kononov (2006) designed a 3.5-approximation algorithms for problem $1|(a_j, l_j, b_j)|C_{\max}$ and proved that a (2-ε) approximation algorithm does not exist unless P=NP. Yu et al. (2004) implied the strong NP-hardness of problem $1|(1, l_j, 1)|C_{\max}$ from the strong NP-hardness proof of problem $F2|(1, l_j, 1)|C_{\max}$. Ageev & Baburin (2007) designed a 7/4-approximation algorithm for problem $1|(1, l_j, 1)|C_{\max}$. Subsequently, Békési et al. (2009) improved the analysis of Ageev & Baburin (2007) to derive a better lower bound of the approximation ratio. Furthermore, Li & Zhao (2007) designed approximation

algorithms for some NP-hard special cases, and developed a tabu search meta-heuristic for the general case.

To the best of our knowledge, the constraint of a fixed job sequence was not previously addressed in the coupled-task problem. For the problem under study, a predetermined job sequence defines a sequence of the first tasks of all jobs and the same sequence of the second tasks of all jobs. To construct a feasible schedule subject to a fixed job sequence, the decision is how to interleave the task-1 sequence and the task-2 sequence. Due to the strong NP-hardness of problem $1|(a_j, l_j, b_j)|C_{\max}$, it is appropriate to study the $1|(a_j, l_j, b_j), fjs|C_{\max}$ problem. Besides the assumption of a fixed job sequence could also be suitable for the practical application. A naval warship is commonly equipped with a radar with the capacities of surveillance, tracking and weapon guidance, which is called a multifunction radar (Orman et al., 1998). A multifunction radar system simultaneously performs the missions of surveillance, tracking and weapon guidance. There exists a well-defined priority structure with these three missions. The mission of weapon guidance retains the highest priority. For the mission of tracking, an inbound target has higher priority than the object moving away. If a precedence chain of the targets of all missions is predetermined, then the assumption of a fixed job sequence in coupled-task scheduling could be reasonable.

In the following section, several intriguing properties of the fixed-job-sequence problem are expounded in detail. While the complexity status of the considered problem remains open, a polynomial-time algorithm is presented to construct a schedule with the minimum makespan for a given task sequence abiding by the fixed-job- sequence constraint. Three polynomially solvable cases of the fixed-job-sequence problem are also investigated and a complexity graph of the problem is presented.

4.2 Problem Description

Without loss of generality, this study assumes that the fixed job sequence is $(J_1, J_2, \ldots, J_n)$. Subject to the constraint of a fixed job sequence and the definition of coupled tasks, we thus have a directed ladder graph of precedence relationship with two long chains,

$a_1 \to a_2 \to \cdots \to a_n$ and $b_1 \to b_2 \to \cdots \to b_n$, and n single-arc chains, $a_j \to b_j$ for all $j \in \mathbb{N}_n$, as illustrated in Figure 4.1. As a permutation of $\{a_j \,|\, j \in \mathbb{N}_n\} \cup \{b_j \,|\, j \in \mathbb{N}_n\}$, a task sequence is called *plausible* if it adheres to the precedence constraints given by the ladder graph. An initial idea about how to generate a plausible task sequence is given first. By virtue of the diagonal-avoiding paths (Davis, 2006), the following observation is presented.

Observation 4.1. *Given n jobs, all plausible task sequences can be generated by the diagonal-avoiding paths along the edges of a grid with $n \times n$ square cells. Each diagonal-avoiding path corresponds to exactly one plausible task sequence.*

A diagonal-avoiding path is the one which leads from the top-left corner O to the bottom-right corner D without backtracking, and stays on or above the diagonal without passing below it. As shown in Figure 4.2 for the case $n = 5$, the illustrated diagonal-avoiding path corresponds to the plausible task sequence $(a_1, a_2, b_1, a_3, b_2, b_3, a_4, a_5, b_4, b_5)$. The number of diagonal-avoiding paths in a grid of $n \times n$ squares is given by the well-known Catalan number, $\mathcal{C}_n = \frac{1}{n+1}\binom{2n}{n}$, which grows in the order of $\Omega(4^n/\sqrt{n^3})$.

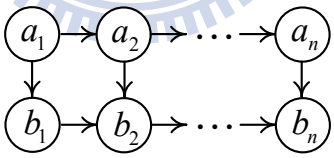


Figure 4.1: A directed ladder graph of precedence relationship for the $1|(a_j, l_j, b_j), fjs|C_{\max}$ problem.

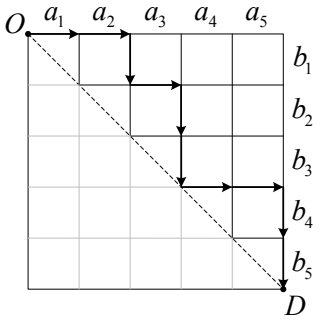


Figure 4.2: The diagonal-avoiding path corresponding to the plausible task sequence $(a_1, a_2, b_1, a_3, b_2, b_3, a_4, a_5, b_4, b_5)$.

Previous experience suggests that scheduling problems with fixed job sequences could be resolved by dynamic programs for the objective of makespan or total completion time minimization (Cheng et al., 2000b; Hwang et al., 2010a; Ng & Kovalyov, 2007). However, it seems not to be the case for problem $1|(a_j, l_j, b_j), fjs|C_{\max}$. The intrigue could be ascribed to the following two causes:

1. Although the job sequence is pre-assigned, the studied problem remains a problem of sequencing in which there are $\frac{1}{n+1}\binom{2n}{n}$ plausible task sequences for n jobs.

2. The time lag l_j between tasks a_j and b_j can accommodate the processing of not only tasks $\{a_{j+1}, a_{j+2}, \ldots, a_n\}$ but also tasks $\{b_1, b_2, \ldots, b_{j-1}\}$. Due to the distinctive scheduling pattern, the principle of optimality fails. Thus, it becomes not clear whether a dynamic programming approach will work. Take for example the following instance with five jobs: $(a_1, l_1, b_1) = (3, 9, 1)$, $(a_2, l_2, b_2) = (1, 10, 2)$, $(a_3, l_3, b_3) = (2, 11, 3)$, $(a_4, l_4, b_4) = (3, 10, 2)$, $(a_5, l_5, b_5) = (3, 9, 1)$. The optimal schedule for the fixed-job-sequence problem is demonstrated in Figure 4.3. In the optimal schedule, the subschedule of the subsequence (J_1, J_2) attains the time span equal to 18. But the time span of the shortest subschedule constructed with the subsequence (J_1, J_2) is 16. Thus, a subschedule within the shortest complete schedule is not necessarily a shortest subschedule.

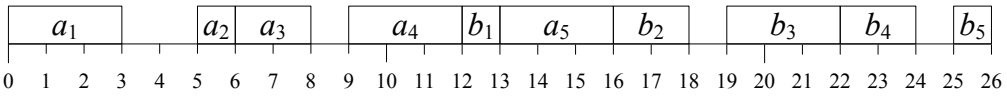


Figure 4.3: Optimal schedule of the instance with $(a_1, l_1, b_1) = (3, 9, 1)$, $(a_2, l_2, b_2) = (1, 10, 2)$, $(a_3, l_3, b_3) = (2, 11, 3)$, $(a_4, l_4, b_4) = (3, 10, 2)$, $(a_5, l_5, b_5) = (3, 9, 1)$.

Owing to the intrigue of the fixed-job-sequence problem, we turn to aim at scheduling a given plausible task sequence in the next section. The notation that will be used later is summarized as follows.

Notation:

π : a given plausible sequence of $2n$ tasks, $\pi = (o_1, o_2, \ldots, o_{2n})$;

o_h : the task assigned to the h-th position in π;

σ : a schedule of $2n$ tasks;

$s_j(\sigma)$: the starting time of job J_j in a schedule σ;

$C_j(\sigma)$: the completion time of job J_j in a schedule σ;

k : the number of segments in the task sequence $(a_1, a_2, \ldots, a_n)$;

X : the sequence of subscripts $(1, 2, \ldots, n)$;

H : a k-subsequence partition of X corresponding to the partition of $(a_1, a_2, \ldots, a_n)$;

X_r : the r-th subsequence in X, $X_r = (n_{r-1} + 1, \ldots, n_r)$, where $n_0 = 0$ and $n_k = n$;

$\tilde{X}_r$: the set of elements in sequence X_r;

$|X_r|$: the length of X_r, $|X_r| = n_r - n_{r-1}$;

$b_{n'_r}$: the immediate predecessor of a_{n_r+1} in π;

π_r : the r-th fundamental cluster in π, $\pi_r = (a_{n_{r-1}+1}, \ldots, a_{n_r}, b_{n_{r-1}+1}, \ldots, b_{n_r})$;

χ_r : The subsequence obtained by eliminating the jobs of $\{J_j \,|\, j \in \tilde{X}_{r+1} \cup \cdots \cup \tilde{X}_k\}$ from π;

σ_r : the schedule of π_r constructed by the developed recursive formula;

σ_{π_r} : a feasible schedule whose permutation of tasks agrees with π_r;

α_r : the time span from the start of $a_{n_{r-1}+1}$ to the completion of a_{n_r};

γ_r : the idle time between a_{n_r} and $b_{n_{r-1}+1}$;

S_r : a subschedule constructed by arranging the first r subschedules, $\sigma_1, \ldots, \sigma_r$;

β_r^1 : the idle time between $b_{n'_r}$ and $b_{n'_r+1}$ in subschedule S_r;

β_r^2 : the time span from the start of $b_{n'_r+1}$ to the completion of b_{n_r};

μ : the input task of **Checking routine** in **Algorithm Plausible-Task-Sequence**;

ν : the task preceded by μ in χ_{I+1};

$\tilde{\nu}$: the corresponding first or second counterpart task of ν.

4.3 Scheduling of Plausible Task Sequences

Discussion in the previous section introduces the notion of $\frac{1}{n+1}\binom{2n}{n}$ plausible task sequence for a given job sequence. This section is dedicated to the development of a polynomial time algorithm for determining the makespan of a plausible task sequence, if it is feasible. Given a plausible task sequence, it could be non-trivial to determine its feasibility and a schedule with the minimum makespan, if feasible.

Denote a plausible task sequence by $\pi = (o_1, o_2, \ldots, o_{2n})$, where o_h stands for the task assigned to the h-th position in π. If no confusion would arise, hereafter this study simply mentions sequences to indicate plausible task sequences. Notice that in any schedule considered hereafter, the constraint of exact delays is satisfied. In other words, the interval between each pair of coupled tasks a_j and b_j in any schedule is exactly l_j, $j \in \mathbb{N}_n$. Denote the starting time and the completion time of job J_j in a schedule σ by $s_j(\sigma)$ and $C_j(\sigma)$, respectively. It is obvious that $C_j(\sigma) = s_j(\sigma) + a_j + l_j + b_j$. A schedule σ is feasible if and only if at any time, at most one task is processed in σ, i.e. no overlap between tasks occurs. Sequence π is called *feasible* if and only if there exists a feasible schedule whose permutation of tasks agrees with π, i.e. a schedule in which the processing of any task o_h for $h \in \mathbb{N}_{2n-1}$ completes earlier than or exactly at the starting time of task o_{h+1}.

Consider first how to determine the feasibility of a given sequence π. For any feasible sequence π, the constraint of exact delays implies that the interval induced by the exact delay l_j of any job J_j must accommodate all the tasks arranged between a_j and b_j. Namely, the following condition is necessary for the feasibility of a sequence π.

Condition (C): For any job J_j with $o_\ell = a_j$ and $o_g = b_j$, the inequality $\sum_{h=\ell+1}^{g-1} p_{o_h} \leq l_j$ must hold, where p_{o_h} is the processing length of task o_h.

Note that condition (C) is not sufficient to make sequence π feasible. Consider the

following instance: $(a_1, l_1, b_1) = (1, 2, 1)$, $(a_2, l_2, b_2) = (2, 5, 1)$, $(a_3, l_3, b_3) = (2, 4, 2)$, $(a_4, l_4, b_4) = (2, 3, 1)$. Condition (C) holds for sequence $\pi = (a_1, a_2, b_1, a_3, a_4, b_2, b_3, b_4)$. However, π is infeasible since an overlap between b_3 and b_4 (Figure 4.4(a)) or between a_4 and b_2 (Figure 4.4(b)) is inevitable in any attempt to create a feasible schedule of π.

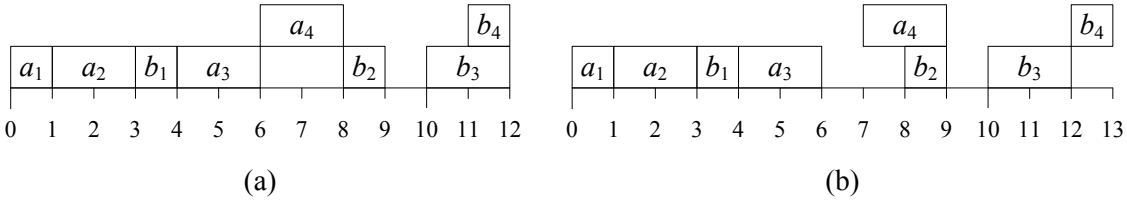


Figure 4.4: An inevitable task overlap happens (a) between b_3 and b_4 or (b) between a_4 and b_2.

Condition (C) only partially verifies the feasibility of π because in a schedule whether an idle time or overlap exists between o_h and o_{h+1} cannot be detected before assigning each task a starting time. Therefore, we turn to develop a procedure for constructing a schedule for sequence π and prove that the feasibility of π can be determined by the constructed schedule. If sequence π is indeed feasible, it can be further proved that the constructed schedule attains the minimum makespan among those of feasible schedules.

Consider the subsequences of a particular permutation pattern

$$(a_{i_1}, a_{i_1+1}, \ldots, a_{i_2}, b_{i_1}, b_{i_1+1}, \ldots, b_{i_2}),$$

$1 \leq i_1 \leq i_2 \leq n$, where all its first (respectively, second) tasks are consecutively sequenced without any second (respectively, first) task inserted. A given sequence π is derived by merging several subsequences of this pattern. Consider the sequence $\pi = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, a_6, b_4, a_7, a_8, b_5, b_6, b_7, b_8)$ as an example. As shown in Figure 4.5, sequence π can be regarded as the outcome of four interleaved subsequences (a_1, a_2, b_1, b_2), $(a_3, a_4, a_5, b_3, b_4, b_5)$, (a_6, b_6) and (a_7, a_8, b_7, b_8). Such particular subsequences are regarded as the maximal *fundamental clusters* of a given sequence π and these subsequences are scheduled individually. Scheduling these fundamental clusters is the first attempt to examine the feasibility of sequence π. Later it will be elucidated that the infeasibility of

any fundamental cluster leads to the infeasibility of π. If all these fundamental clusters are feasible, then we proceed to schedule π by interleaving those obtained subschedules.

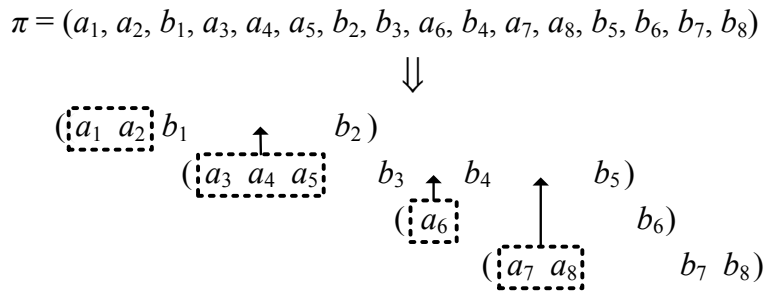


Figure 4.5: Four subsequences of a particular pattern in sequence π.

Now some notations for collating fundamental clusters from sequence π are defined. Given a sequence π, a *segment* is defined as a maximal, by inclusion, subsequence of tasks $\{a_j\}$ without inserted tasks $\{b_i\}$. Assume that the task sequence $(a_1, a_2, \ldots, a_n)$ is partitioned into k disjoint segments for $1 \leq k \leq n$. To facilitate discussion, we denote by X the sequence of subscripts $(1, 2, \ldots, n)$ and by H a k-subsequence partition of X corresponding to the k-segment partition of $(a_1, a_2, \ldots, a_n)$. Partitioning X into k disjoint subsequences, we have $H = \{X_1, X_2, \ldots, X_k\}$ and $X = X_1 \oplus X_2 \oplus \cdots \oplus X_k$, where X_r denotes the r-th subsequence in X and $\oplus$ is a sequence concatenation operator. For $r \in \mathbb{N}_k$, the last element of subsequence X_r is denoted by n_r, and we have $X_r = (n_{r-1} + 1, \ldots, n_r)$, where $n_0 = 0$ and $n_k = n$. Denote by $\tilde{X}_r$ the set of elements in sequence X_r. $|X_r| = n_r - n_{r-1}$ indicates the length of X_r. Denote by $b_{n'_r}$ the immediate predecessor of a_{n_r+1} in π for $r \in \mathbb{N}_{k-1}$ and $n'_{r-1} + 1 \leq n'_r \leq n_r$, where $n'_0 = 0$. Notice that any single task $a_{n_{r-1}+1} = a_{n_r}$, which is surrounded by two tasks $b_{n'_{r-1}}$ and $b_{n'_{r-1}+1}$ in π, forms a segment, i.e. $|X_r| = 1$. According to the assumption of k segments, sequence π consists of k fundamental clusters in which the r-th one contains jobs $\{J_j \,|\, j \in \tilde{X}_r\}$, $r \in \mathbb{N}_k$. Denote the r-th fundamental cluster in π by $\pi_r = (a_{n_{r-1}+1}, \ldots, a_{n_r}, b_{n_{r-1}+1}, \ldots, b_{n_r})$, $r \in \mathbb{N}_k$. The subsequence obtained by eliminating the jobs of $\{J_j \,|\, j \in \tilde{X}_{r+1} \cup \cdots \cup \tilde{X}_k\}$ from π is denoted by χ_r, $r \in \mathbb{N}_{k-1}$. Note that $\chi_k = \pi$.

To construct a schedule of fundamental cluster π_r, we proposes a recursive procedure to augment the subschedule job by job, instead of task by task. Namely, coupled tasks a_j

and b_j are simultaneously added into the subschedule of jobs $(J_{n_{r-1}+1}, J_{n_{r-1}+2}, \ldots, J_{j-1})$, $j \in \{n_{r-1}+2, \ldots, n_r\}$, in each recursion step. In the proposed procedure, job J_j is interleaved with job J_{j-1} by conjoining two first tasks a_{j-1} and a_j (Figure 4.6(a)) or two second tasks b_{j-1} and b_j (Figure 4.6(b)). The obtained schedule is denoted by σ_r, and the recursive formula for the job starting times is given as follows:

$$s_j(\sigma_r) = \begin{cases} 0, & j = n_{r-1}+1; \\ s_{j-1}(\sigma_r) + a_{j-1} & \\ \quad + \max\{0, l_{j-1} + b_{j-1} - a_j - l_j\}, & n_{r-1}+2 \le j \le n_r. \end{cases} \tag{4.1}$$

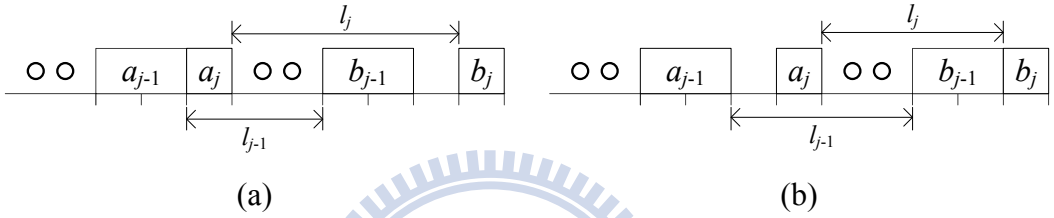


Figure 4.6: J_j is interleaved with J_{j-1} by conjoining (a) a_{j-1} and a_j or (b) b_{j-1} and b_j.

Eq. (4.1) implies that in σ_r task a_j (respectively, b_j) is started later than or exactly at the completion of a_{j-1} (respectively, b_{j-1}). Schedule σ_r is a feasible schedule of π_r if task a_{n_r} completes earlier than or exactly at the start of task $b_{n_{r-1}+1}$. The following lemma gives structural properties of fundamental clusters.

Lemma 4.1. *Given a subsequence $\pi_r = (a_{n_{r-1}+1}, \ldots, a_{n_r}, b_{n_{r-1}+1}, \ldots, b_{n_r})$, the following three properties hold: (i) If $s_{n_r}(\sigma_r) + a_{n_r} > C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$, then π_r is infeasible. (ii) If $s_{n_r}(\sigma_r) + a_{n_r} \le C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$, then π_r is feasible and σ_r is a feasible schedule attaining the minimum makespan among those of all feasible schedules of π_r. (iii) The feasibility and the shortest schedule, if feasible, can be determined in $O(|X_r|)$ time.*

Proof. If $s_{n_r}(\sigma_r) + a_{n_r} > C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$, then task a_{n_r} completes later than the start of task $b_{n_{r-1}+1}$ in σ_r. The only possible way to find a feasible schedule of π_r is to process a_{n_r} earlier or process $b_{n_{r-1}+1}$ later. In schedule σ_r, task a_j starts exactly at the completion of a_{j-1}, or task b_j starts exactly at the completion of b_{j-1}, $j \in \{n_{r-1}+2, \ldots, n_r\}$. Starting J_{n_r} earlier or $J_{n_{r-1}+1}$ later finally results in the shifting of the whole schedule, which is

futile. Therefore, a feasible schedule of π_r does not exist and π_r is infeasible. Property (i) is proved.

Property (ii) is concerned about the feasibility of π_r and the optimality of σ_r. As for the feasibility of π_r, the inequality $s_{n_r}(\sigma_r) + a_{n_r} \leq C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$ indicates that task a_{n_r} completes earlier than or exactly at the starting time of task $b_{n_{r-1}+1}$ in σ_r. Therefore, a feasible schedule of π_r, i.e. σ_r, exists and π_r is feasible. Next, we will show that schedule σ_r attains the minimum makespan for sequence π_r, i.e. $C_{n_r}(\sigma_r) \leq C_{n_r}(\sigma_{\pi_r})$, where σ_{π_r} denotes any feasible schedule of π_r. This inequality can be proved by induction on n_r. We derive the following recursive equation for $C_j(\sigma_r)$ by adapting Eq. (4.1).

$$C_j(\sigma_r) = \begin{cases} a_{n_{r-1}+1} + l_{n_{r-1}+1} + b_{n_{r-1}+1}, & j = n_{r-1}+1; \\ C_{j-1}(\sigma_r) + b_j & \\ + \max\{0, a_j + l_j - l_{j-1} - b_{j-1}\}, & n_{r-1}+2 \leq j \leq n_r. \end{cases} \tag{4.2}$$

Consider the induction base $n_r = n_{r-1}+2$ and $\pi_r = (a_{n_{r-1}+1}, a_{n_{r-1}+2}, b_{n_{r-1}+1}, b_{n_{r-1}+2})$. If $a_{n_{r-1}+2} + l_{n_{r-1}+2} > l_{n_{r-1}+1} + b_{n_{r-1}+1}$, then it is impossible to interleave $J_{n_{r-1}+1}$ and $J_{n_{r-1}+2}$ with a completion time less than $a_{n_{r-1}+1} + a_{n_{r-1}+2} + l_{n_{r-1}+2} + b_{n_{r-1}+2}$. On the other hand, if $a_{n_{r-1}+2} + l_{n_{r-1}+2} \leq l_{n_{r-1}+1} + b_{n_{r-1}+1}$, then there exists no feasible schedule σ_{π_r} where $J_{n_{r-1}+2}$ completes earlier than $a_{n_{r-1}+1} + l_{n_{r-1}+1} + b_{n_{r-1}+1} + b_{n_{r-1}+2}$. Since $C_{n_{r-1}+2}(\sigma_r) = a_{n_{r-1}+1} + a_{n_{r-1}+2} + l_{n_{r-1}+2} + b_{n_{r-1}+2}$ for $a_{n_{r-1}+2} + l_{n_{r-1}+2} > l_{n_{r-1}+1} + b_{n_{r-1}+1}$ and $C_{n_{r-1}+2}(\sigma_r) = a_{n_{r-1}+1} + l_{n_{r-1}+1} + b_{n_{r-1}+1} + b_{n_{r-1}+2}$ for $a_{n_{r-1}+2} + l_{n_{r-1}+2} \leq l_{n_{r-1}+1} + b_{n_{r-1}+1}$ from Eq. (4.2), we have the induction base $C_{n_{r-1}+2}(\sigma_r) \leq C_{n_{r-1}+2}(\sigma_{\pi_r})$.

Assume, as the induction hypothesis, that the inequality holds for $n_r = i > n_{r-1}+2$, i.e. $C_i(\sigma_r) \leq C_i(\sigma_{\pi_r})$. If $a_{i+1} + l_{i+1} > l_i + b_i$, then the minimum time span from the completion time of J_i to that of J_{i+1} is $a_{i+1} + l_{i+1} + b_{i+1} - l_i - b_i$. In case of $a_{i+1} + l_{i+1} \leq l_i + b_i$, the minimum aforementioned time span is b_{i+1}. By Eq. (4.2), we have $C_{i+1}(\sigma_r) = C_i(\sigma_r) + b_{i+1} + a_{i+1} + l_{i+1} - l_i - b_i$ for $a_{i+1} + l_{i+1} > l_i + b_i$ and $C_{i+1}(\sigma_r) = C_i(\sigma_r) + b_{i+1}$ for $a_{i+1} + l_{i+1} \leq l_i + b_i$. Hence, $C_{i+1}(\sigma_r)$ is less than or equal to the completion time of J_{i+1} in any feasible schedule σ_{π_r}, i.e. $C_{i+1}(\sigma_r) \leq C_{i+1}(\sigma_{\pi_r})$. By induction, the inequality $C_{n_r}(\sigma_r) \leq C_{n_r}(\sigma_{\pi_r})$ is established. The proof of property (ii) is done.

For schedule π_r, the feasibility and the shortest schedule, if feasible, can be obtained by the values of $s_j(\sigma_r)$ for $j = n_{r-1}+1, \ldots, n_r$. By virtue of Eq. (4.1), the calculation involves $|X_r| - 1$ iterations, each of which requires constant time. Therefore, either a feasible schedule with the minimum makespan or the infeasibility of sequence π_r can be determined in $O(|X_r|)$ time. □

The infeasibility of fundamental cluster π_r implies that there exists no feasible subschedule whose task permutation agrees with subsequence π_r, $r \in \mathbb{N}_k$. Since the subsequence π_r is a part of complete sequence π, no feasible complete schedule of π exists either. We therefore have the following property.

Property 4.1. *If any fundamental cluster π_r, $r \in \mathbb{N}_k$, is infeasible, then sequence π is infeasible.*

Before presenting a step-wise procedure for determining the infeasibility or the shortest feasible schedule of sequence π, we define some notations. For each schedule σ_r ($r \in \mathbb{N}_k$), denote by α_r the time span from the start of $a_{n_{r-1}+1}$ to the completion of a_{n_r}, and γ_r the idle time between a_{n_r} and $b_{n_{r-1}+1}$. S_r denotes a subschedule constructed by arranging the first r subschedules, $\sigma_1, \ldots, \sigma_r$. Note that S_k is the constructed complete schedule of sequence π, which consists of k clusters. Denote by β_r^1 the idle time between $b_{n'_r}$ and $b_{n'_r+1}$ in subschedule S_r, and by β_r^2 the time span from the start of $b_{n'_r+1}$ to the completion of b_{n_r}.

Algorithm Plausible-Task-Sequence

Step 1. Scan π to obtain the partition $H = (X_1, \ldots, X_k)$ and keep track of n_r, n'_r and χ_r for each $r = 1, \ldots, k-1$.

Step 2. Schedule π_r by Eq. (4.1) for each $r = 1, \ldots, k$. If any π_r is infeasible, then go to **Step 7**. Otherwise, set $I = 1$ and $S_I = \sigma_1$.

Step 3. If $n'_I = n_I$, then merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of b_{n_I} and go to **Step 6**.

Step 4. If $\beta_I^1 \geq \alpha_{I+1} \wedge \beta_I^2 \leq \gamma_{I+1}$, then go to **Step 4(a)**. If $\beta_I^1 < \alpha_{I+1} \wedge \beta_I^2 \leq \gamma_{I+1}$, then go to **Step 4(b)**. If $\beta_I^1 \geq \alpha_{I+1} \wedge \beta_I^2 > \gamma_{I+1}$, then go to **Step 4(c)**. Otherwise, go to **Step 4(d)**.

Step 4(a). If $\beta_I^1 + \beta_I^2 \leq \alpha_{I+1} + \gamma_{I+1}$, then merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of $b_{n'_I}$. Otherwise, merge S_I with σ_{I+1} by appending b_{n_I+1} to the end of b_{n_I}. Go to **Step 6**.

Step 4(b). Shift $b_{n'_I+1}$ (and $a_{n'_I+1}$ will be simultaneously shifted, i.e. shift $J_{n'_I+1}$) to extend β_I^1 such that $\beta_I^1 = \alpha_{I+1}$ and merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of $b_{n'_I}$. Go to **Step 5**.

Step 4(c). Shift $J_{n'_I+1}$ to shorten β_I^2 such that $\beta_I^2 = \gamma_{I+1}$ and merge S_I with σ_{I+1} by appending b_{n_I+1} to the end of b_{n_I}. Go to **Step 5**.

Step 4(d). If $\beta_I^1 + \beta_I^2 \leq \alpha_{I+1} + \gamma_{I+1}$, then shift $J_{n'_I+1}$ to extend β_I^1 such that $\beta_I^1 = \alpha_{I+1}$ and merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of $b_{n'_I}$. Otherwise, shift $J_{n'_I+1}$ to shorten β_I^2 such that $\beta_I^2 = \gamma_{I+1}$ and merge S_I with σ_{I+1} by appending b_{n_I+1} to the end of b_{n_I}. Go to **Step 5**.

Step 5. Call **Checking routine** with input $b_{n'_I+1}$. Call **Checking routine** with input $a_{n'_I+1}$.

Step 6. Let S_{I+1} be the obtained schedule. If $I = k-1$, then output the schedule S_{I+1} and **stop**. Otherwise, set $I = I+1$ and go to **Step 3**.

Step 7. Report the infeasibility of π and **stop**.

Checking routine. Denote the input task as μ, the task preceded by μ in χ_{I+1} as ν, and the corresponding first or second counterpart task of ν as $\tilde{\nu}$. If $\nu = b_1 \vee a_{n_I+1} \vee b_{n_I+1}$, then go to **Final checking**. Otherwise, go to **Checking and shifting**.

Final checking: If the completion of μ is later than the start of ν, then go to **Step 7**. Otherwise, terminate the subroutine.

Checking and shifting: If the completion of μ is later than the start of ν, shift ν (and $\tilde{\nu}$ will be simultaneously shifted) such that the task ν starts at exactly the completion of μ, call **Checking routine** with input ν, and call again **Checking routine** with input $\tilde{\nu}$. Otherwise, terminate the subroutine.

Example. Consider an instance of eight jobs with the following parameters (Figure 4.7(a)): $(a_1, l_1, b_1) = (3, 4, 1)$, $(a_2, l_2, b_2) = (1, 7, 2)$, $(a_3, l_3, b_3) = (1, 8, 1)$, $(a_4, l_4, b_4) = (1, 10, 1)$, $(a_5, l_5, b_5) = (2, 9, 1)$, $(a_6, l_6, b_6) = (1, 4, 3)$, $(a_7, l_7, b_7) = (1, 5, 2)$, $(a_8, l_8, b_8) = (1, 6, 1)$. The sequence $\pi = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, a_6, b_4, a_7, a_8, b_5, b_6, b_7, b_8)$ is given. Constructing a feasible schedule σ_π attaining the minimum makespan with **Algorithm Plausible-Task-Sequence** is demonstrated step by step as follows:

Step 1. We obtain $k = 4$, $X_1 = (1, 2)$, $X_2 = (3, 4, 5)$, $X_3 = (6)$, $X_4 = (7, 8)$, $n_1 = 2$, $n_2 = 5$, $n_3 = 6$, $n'_1 = 1$, $n'_2 = 3$, $n'_3 = 4$, $\chi_1 = (a_1, a_2, b_1, b_2)$, $\chi_2 = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, b_4, b_5)$, and $\chi_3 = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, a_6, b_4, b_5, b_6)$.

Step 2. Feasible subschedules $\sigma_1, \sigma_2, \sigma_3$ and σ_4 are derived as shown in Figure 4.7(b). Set $I = 1$ and $S_1 = \sigma_1$.

Step 3. Since $n'_1 = 1 < n_1 = 2$, we go to **Step 4**.

Step 4. Since $\beta_1^1 = 3 < \alpha_2 = 4$ and $\beta_1^2 = 2 < \gamma_2 = 5$, we go to **Step 4(b)**.

Step 4(b). Shift J_2 such that $\beta_1^1 = \alpha_2 = 4$ and merge S_1 with σ_2 by appending a_3 to the end of b_1. Go to **Step 5**.

Step 5. Call **Checking routine** with input b_2. Call **Checking routine** with input a_2.

Checking routine. We have $\mu = b_2$, $\nu = b_3$, and $\tilde{\nu} = a_3$. Go to **Final checking**.

Final checking: Task b_2 completes (at 14) earlier than the start of b_3 (at 17). Terminate the subroutine.

Checking routine. We have $\mu = a_2$, $\nu = b_1$, and $\tilde{\nu} = a_1$. Go to **Final checking**.

Final checking: Task a_2 completes (at 5) earlier than the start of b_1 (at 7). Terminate the subroutine.

Step 6. The obtained schedule is S_2 (Figure 4.8). Set $I = 2$ and go to **Step 3**.

Step 3. Since $n'_2 = 3 < n_2 = 5$, we go to **Step 4**.

Step 4. Since $\beta_2^1 = 2 > \alpha_3 = 1$ and $\beta_2^2 = 2 < \gamma_3 = 4$, we go to **Step 4(a)**.

Step 4(a). Since $\beta_2^1 + \beta_2^2 = 4 < \alpha_3 + \gamma_3 = 5$, we merge S_2 with σ_3 by appending a_6 to the end of b_3. Go to **Step 6**.

Step 6. The obtained schedule is S_3 (Figure 4.9). Set $I = 3$ and go to **Step 3**.

Step 3. Since $n'_3 = 4 < n_3 = 6$, we go to **Step 4**.

Step 4. Since $\beta_3^1 = 0 < \alpha_4 = 2$ and $\beta_3^2 = 5 > \gamma_4 = 4$, we go to **Step 4(d)**.

Step 4(d). Since $\beta_3^1 + \beta_3^2 = 5 < \alpha_4 + \gamma_4 = 6$, we shift J_5 such that $\beta_3^1 = \alpha_4 = 2$ and merge S_3 with σ_4 by appending a_7 to the end of b_4, as shown in Figure 4.10(a). Go to **Step 5**.

Step 5. Call **Checking routine** with input b_5. Call again **Checking routine** with input a_5.

Checking routine. We have $\mu = b_5$, $\nu = b_6$, and $\tilde{\nu} = a_6$. Go to **Checking and shifting**.

Checking and shifting: Task b_5 completes (at 24) later than the start of b_6 (at 23). Shift J_6 such that the start of b_6 is exactly at 24, as shown in Figure 4.10(b). Call **Checking routine** with input b_6, and call again **Checking routine** with input a_6.

Checking routine. We have $\mu = b_6$, $\nu = b_7$, and $\tilde{\nu} = a_7$. Go to **Final checking**.

Final checking: Task b_6 completes (at 27) exactly at the start of b_7. Terminate the subroutine.

Checking routine. We have $\mu = a_6$, $\nu = b_4$, and $\tilde{\nu} = a_4$. Go to **Checking and shifting**.

Checking and shifting: Task a_6 completes (at 20) exactly at the start of b_4. Terminate the subroutine.

Checking routine. We have $\mu = a_5$, $\nu = b_2$, and $\tilde{\nu} = a_2$. Go to **Checking and shifting**.

Checking and shifting: Task a_5 completes (at 14) later than the start of b_2 (at 12). Shift J_2 such that the start of b_2 is exactly at 14, as shown in Figure 4.10(c). Call **Checking routine** with input b_2, and call again **Checking routine** with input a_2.

Checking routine. We have $\mu = b_2$, $\nu = b_3$, and $\tilde{\nu} = a_3$. Go to **Checking and shifting**.

Checking and shifting: Task b_2 completes (at 16) earlier than the start of b_3 (at 17). Terminate the subroutine.

Checking routine. We have $\mu = a_2$, $\nu = b_1$, and $\tilde{\nu} = a_1$. Go to **Final checking**.

Final checking: Task a_2 completes (at 7) exactly at the start of b_1. Terminate the subroutine.

Step 6. The obtained schedule is S_4. Since $I = 3 = k - 1$, we output the schedule S_4 (Figure 4.10(c)) and **stop**.

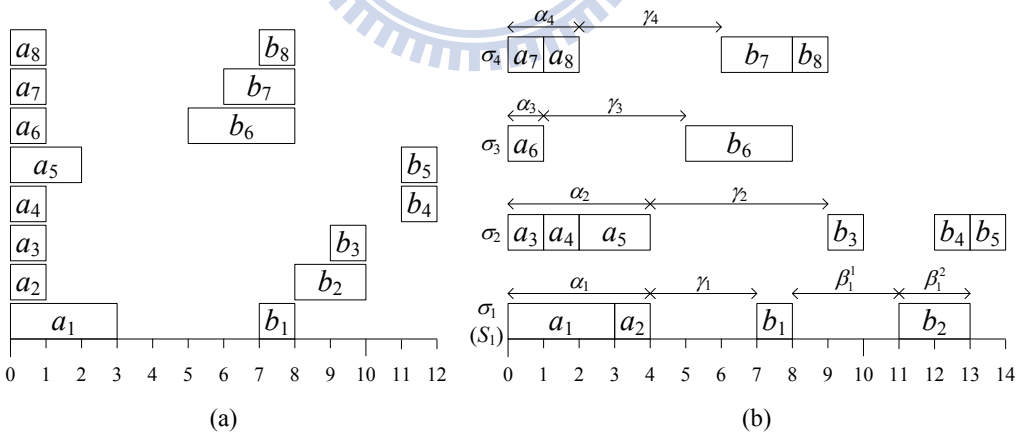


Figure 4.7: Illustration of (a) the instance of eight jobs and (b) subschedules $\sigma_1(= S_1)$, σ_2, σ_3 and σ_4.

Theorem 4.1. *Given a sequence π, **Algorithm Plausible-Task-Sequence** either produces a feasible schedule attaining the minimum makespan or identifies the infeasibility of π in $O(n^2)$ time.*

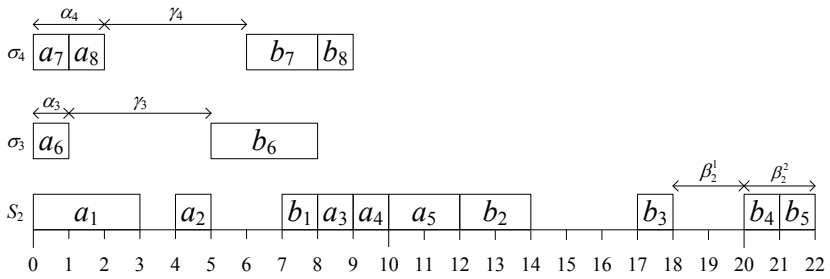


Figure 4.8: Subschedules S_2, σ_3 and σ_4.

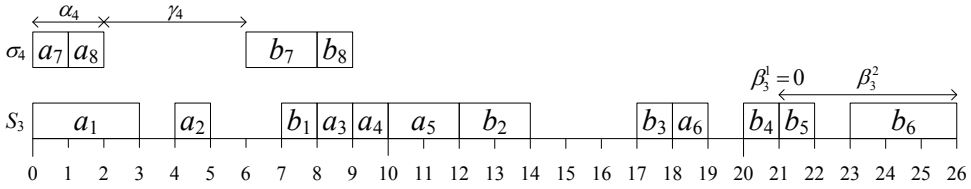


Figure 4.9: Subschedules S_3 and σ_4.

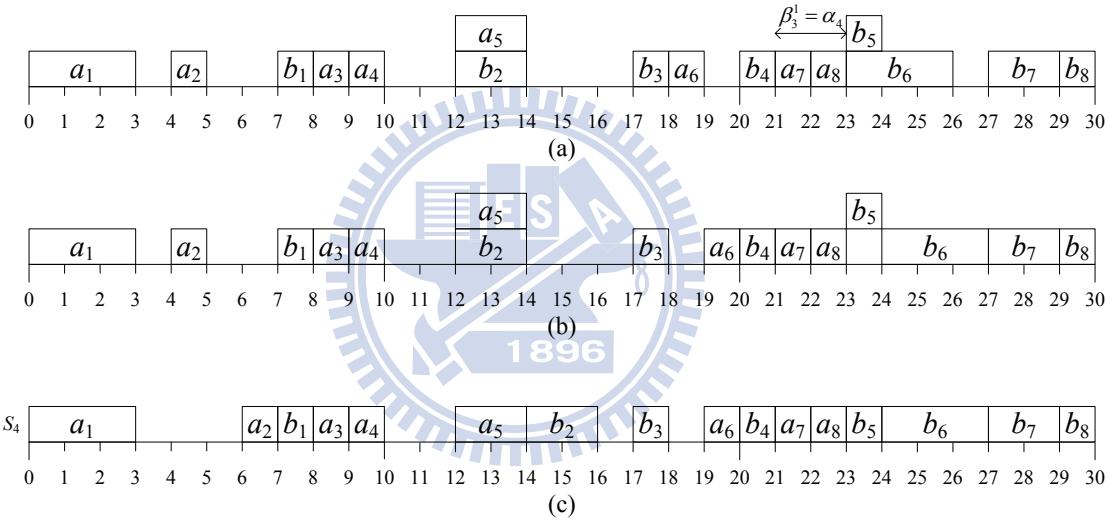


Figure 4.10: Step-by-step construction of optimal schedule S_4.

Proof. Assume a feasible schedule S_k is produced by the algorithm. At the end of **Step 2**, we have the k subschedules, $\sigma_1, \ldots, \sigma_k$, each of which attains the minimum makespan with respect to its corresponding fundamental cluster. In the recursive procedure from **Step 3** to **Step 6**, S_k is obtained by tightly arranging all the k partial schedules, $\sigma_1, \ldots, \sigma_k$, one by one. In **Step 3**, we can easily merge S_I with σ_{I+1} without shifting any task of S_I because $b_{n'_I}$, the task by which a_{n_I+1} should be preceded, is known to be b_{n_I}. In case of **Step 4(a)**, σ_{I+1} can also be greedily embedded in S_I without shifting any task of S_I because $\beta_I^1 \geq \alpha_{I+1}$ and $\beta_I^2 \leq \gamma_{I+1}$. Notice that in **Steps 4(b)-(d)**, it is required to defer the processing of $J_{n'_I+1}$, but all jobs other than $J_{n'_I+1}$ are not yet shifted. **Step 5** involves

calling **Checking routine**s with the two tasks $b_{n'_I+1}$ and $a_{n'_I+1}$, respectively. Whenever **Checking routine** is invoked, either **Final checking** or **Checking and shifting** will be executed. Subroutine **Final checking** indicates that the infeasible result can be concluded whenever task b_1 (also, a_1), a_{n_I+1} or b_{n_I+1} needs to be shifted. In subroutine **Checking and shifting**, we examine whether any job needs to be shifted due to the shifting of the task passed to **Checking routine**. If the shifting of other tasks are made, then **Checking routine** will be called again. Provided that a feasible schedule S_k is obtained after the recursive procedure, either the first or second task of the job J_j in S_k tightly adjoins the task preceding it in π, for each $j = 2, \ldots, n$. No room is possible to further condense S_k.

Consider the case of infeasibility. If the infeasibility of π arises from **Step 2**, then it is due to the results of Property 4.1. If infeasible comes from the subroutine **Final checking**, then some task completes later than the start of b_1, a_{n_I+1} or b_{n_I+1}. It is obvious that shifting J_1 is futile and shifting J_{n_I+1} causes an infinite shifting recursion. Therefore, sequence π is infeasible if infeasibility is reported by the algorithm.

As for the running time of the algorithm, **Step 1** requires $O(n)$ time. **Step 2** takes at most $O(|X_1| + |X_2| + \cdots + |X_k|) = O(n)$ time, and the recursion from **Step 3** to **Step 6** involves assembling k partial schedules, each of which takes no more than $O(n)$ time for the checking processes in **Step 5**. Since $k \leq n$, the overall running time is $O(n^2)$. □

4.4 Polynomially Solvable Cases

This section discusses three polynomially solvable cases for the fixed-job-sequence problem. Notice that the complexity result in this section is presented subject to the assumption of input size such that, for example, in the case of identical jobs, we have n copies of processing times and delay times for the n jobs (Orman & Potts, 1997).

4.4.1 $1|(p_j, p_j, p_j), fjs|C_{\max}$

In this subsection, the case where $a_j = l_j = b_j = p_j$ for all $j \in \mathbb{N}_n$ is considered. Despite the strong NP-hardness of the $1|(p_j, p_j, p_j)|C_{\max}$ problem (Orman & Potts, 1997), its fixed-job-sequence counterpart is polynomially solvable. An optimal schedule can be obtained by the following procedure.

Algorithm PSC1

Step 1. Set $j = 1$ and $E = \emptyset$.

Step 2. If $p_j = p_{j+1}$, then go to **Step 4**. Otherwise, append J_{j+1} to the end of J_j, and set $j = j + 1$.

Step 3. If $j = n$, then output the schedule and **stop**. Otherwise, go to **Step 2**.

Step 4. Interleave J_j and J_{j+1}. Append J_{j+2} to the end of J_{j+1}. Set $E = E \cup \{j, j+1\}$ and $j = j + 2$. Go to **Step 3**.

Theorem 4.2. *The $1|(p_j, p_j, p_j), fjs|C_{\max}$ problem can be solved in $O(n)$ by **Algorithm PSC1**. The makespan of the optimal schedule is $2\sum_{j\in E} p_j + 3\sum_{j\in\mathbb{N}_n\setminus E} p_j$, where E is the set of jobs interleaving with each other.*

Proof. It is obvious that no interleaving is possible for any two jobs other than two adjacent identical jobs, J_j and J_{j+1} with $p_j = p_{j+1}$. By examining each pair of adjacent jobs, **Algorithm PSC1** matches any un-interleaved J_j with J_{j+1} if $p_j = p_{j+1}$. Since no more interleaving is possible, **Algorithm PSC1** produces an optimal schedule. In the obtained optimal schedule, each interleaved pair of jobs, J_j and J_{j+1} for $\{j, j+1\} \subset E$, contributes $2(p_j + p_{j+1})$ to the makespan. Any job J_h that cannot be interleaved contributes $3p_h$ to the makespan. Thus, $C_{\max} = 2\sum_{j\in E} p_j + 3\sum_{j\in\mathbb{N}_n\setminus E} p_j$. From **Step 2** to **Step 4**, at most n iterations are required, each of which takes a constant time. The overall running time of **Algorithm PSC1** is $O(n)$. □

4.4.2 $1|(p,p,b_j),fjs|C_{\max}$

Without the assumption of a fixed job sequence, this special case can be solved in $O(n)$ time (Orman & Potts, 1997). Since $a_j = l_j = p$ for $j \in \mathbb{N}_n$, any job cannot be interleaved with more than one job. Subject to a given job sequence, we have the following property of this special case.

Property 4.2. *If the interleaving of jobs exists in a feasible schedule for the problem* $1|(p,p,b_j),fjs|C_{\max}$*, then the interleaved pair are some two consecutive jobs* J_j *and* J_{j+1}*, where* $b_j \leq p$*,* $j \in \mathbb{N}_{n-1}$*.*

With property 4.2, a forward dynamic program can be designed. A job is called isolated if it is not interleaved with any other job. A subschedule of $\{J_1, J_2, \ldots, J_j\}$ can be completely characterized by the 2-tuples (j,λ), where j and λ are the number of jobs in the subschedule and the interleaving status of job J_j, respectively. If $\lambda = 0$, then job J_j is isolated. If $\lambda = 1$, then job J_j is interleaved with job J_{j-1}. Denote the corresponding minimum makespan as $f(j,\lambda)$ for $1 \leq j \leq n$ and $\lambda \in \{0,1\}$.

Algorithm PSC2

Initialization: $f(1,0) = 2p + b_1$ and $f(1,1) = \infty$.

Recursive function: For $2 \leq j \leq n$,

$$f(j,0) = \min\{f(j-1,0), f(j-1,1)\} + 2p + b_j. \tag{4.3}$$

$$f(j,1) = \begin{cases} f(j-1,0) + p + b_j - b_{j-1}, & b_{j-1} \leq p; \\ \infty, & otherwise. \end{cases} \tag{4.4}$$

Goal: $\min_{\lambda \in \{0,1\}} f(n,\lambda)$.

Theorem 4.3. *An optimal schedule for the* $1|(p,p,b_j),fjs|C_{\max}$ *problem can be produced in* $O(n)$ *by* ***Algorithm PSC2****.*

Proof. Eq. (4.3) indicates that any isolated job J_j adjoins J_{j-1} which is either isolated or interleaved with J_{j-2}. In Eq. (4.4), job J_j can be interleaved with job J_{j-1} if J_{j-1} is

isolated and $b_{j-1} \leq p$. A subschedule in the state (j, λ) with value $f(j, \lambda)$ dominates all other subschedules in the same state in the sense that it contributes the minimum value to the makespan among those of all subschedules in this state. The principle of optimality holds and **Algorithm PSC2** can generate an optimal schedule. To obtain $\min_\lambda f(n, \lambda)$, at most $n-1$ iterations are required, each of which takes a constant time. The overall running time of **Algorithm PSC2** is $O(n)$. □

Corollary 4.1. *The* $1|(a_j, p, p), fjs|C_{\max}$ *problem is solvable in* $O(n)$.

Proof. Orman & Potts (1997) proved that the coupled-task makespan problem and its reverse are equivalent. Given the fixed-job-sequence constraint, the equivalence still holds. By virtue of lemma 4.3, this corollary follows. □

4.4.3 $1|(p, l, p), fjs|C_{\max}$

Since all jobs are identical, any feasible schedule for problem $1|(p, l, p)|C_{\max}$ satisfies the fixed-job-sequence constraint. By the results of Orman & Potts (1997) for problem $1|(p, l, p)|C_{\max}$, the fixed-job-sequence problem $1|(p, l, p), fjs|C_{\max}$ can be solved in $O(n)$.

By virtue of these three polynomially solvable cases, we can put the borderline between polynomially solvable problems and open problems in the complexity graph. In correspondence with the complexity graph of the coupled-task scheduling problems shown in Figure 4.11, that of the fixed-job-sequence problems is given in Figure 4.12. The strongly NP-hard problem $1|(p_j, p_j, p_j)|C_{\max}$ becomes polynomially solvable when the fixed-job-sequence assumption is imposed. For each polynomially solvable case of the prototypical problem, the corresponding fixed-job-sequence problem is also solvable in $O(n)$ time. However, it cannot be concluded that a fixed-job-sequence problem is easier to deal with than its counterpart problem without the fixed-job-sequence assumption.

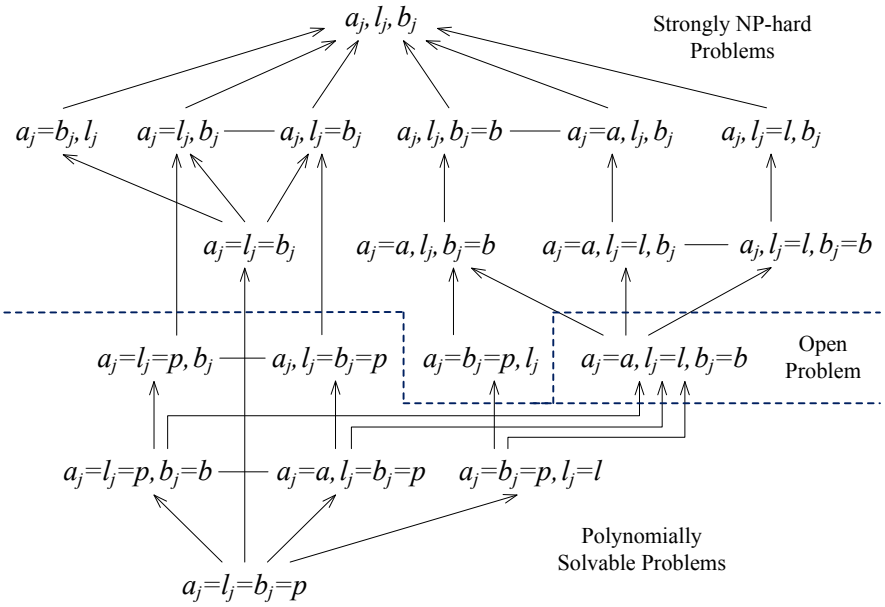


Figure 4.11: Complexity graph of prototypical problems (Orman & Potts, 1997).

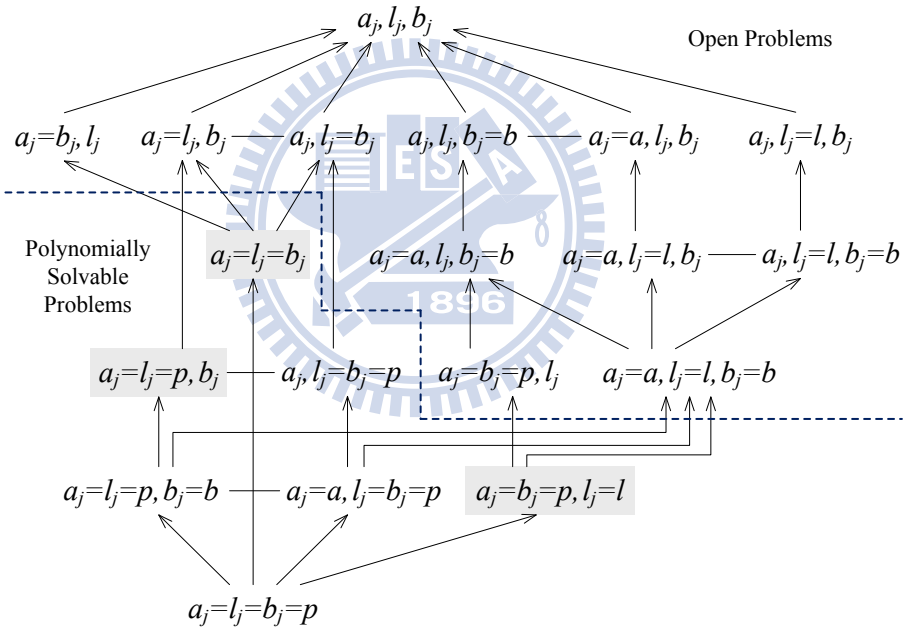


Figure 4.12: Complexity graph of fixed-job-sequence problems.

4.5 Summary

This chapter has studied a single machine coupled-task makespan minimization problem subject to a fixed job sequence. To schedule a given task sequence abiding by the fixed-job-sequence constraint, this study designed an $O(n^2)$ algorithm for determining its feasibility and a schedule with the minimum makespan, if such a feasible schedule exists. Three polynomially solvable cases for the fixed-job-sequence problem were identified. A

complexity graph was also presented to depict the complexity statuses of the studied cases.

Although the complexity status of the $1|(a_j, l_j, b_j), fjs|C_{\max}$ problem remains open, the results presented in this study could inspire further research attention on this subject. It is also interesting to investigate the complexity status of the open problems indicated in the complexity graph of the fixed-job-sequence problems. Further research could also be conducted in developing branch-and-bound procedures in which our proposed algorithm for plausible task sequences could be exploited. In addition, other different fixed-sequence constraints, e.g. given a fixed task-1 sequence or a fixed task-2 sequence, can be considered.

Chapter 5

Concluding Remarks

5.1 Conclusions

In this dissertation, scheduling problems subject to fixed job sequences were studied. The fixed-job-sequence scheduling problems can be categorized into three major types, i.e. problems of batching, problems of interleaving, and problems of idle time insertion. Three fixed-job-sequence scheduling problems, including the two-stage assembly-type flowshop batching problem, the two-stage differentiation flowshop problem and the single-machine coupled-task problem, were studied in this dissertation. The first is concerned with the minimization of total completion time in the two-stage assembly-type flowshop batching problem with a fixed job sequence. A two-phase algorithm equipped with a problem-transformation procedure and a polynomial time dynamic program was developed to solve the studied problem in $O(mn + n^5)$ time, where m is the number of parallel dedicated machines arranged at stage 1 and n is the number of jobs. The second problem is to minimize the total completion time in a two-stage differentiation flowshop subject to fixed sequences of jobs per type. One stage-1 common machine and m stage-2 parallel dedicated machines are arranged in the considered two-stage differentiation flowshop. To achieve an optimal interleaved processing sequence of all jobs at the first stage, we presented an $O(m^2 \prod_{l=1}^{m} n_l^{m+1})$ dynamic programming algorithm, where n_l is the number of type-l jobs. In the third problem, the single-machine coupled-task scheduling with exact delays was investigated. The aim is to schedule these coupled tasks to minimize the makespan

subject to a given job sequence. Several intriguing properties of the studied problem were introduced. While the complexity status of the fixed-job-sequence problem remains open, an $O(n^2)$ algorithm was proposed to construct a feasible schedule attaining the minimum makespan for a given permutation of $2n$ tasks abiding by the fixed-job-sequence constraint. We identified three polynomially solvable cases of the fixed-job-sequence problem and presented a complexity graph of the studied problem.

5.2 Suggestions for Further Studies

Several possible research issues could be developed for further research. Among the fixed-job-sequence scheduling problems, there could be another type of problems except the three categories presented in this study. Also, there could be numerous fixed-job-sequence problems of the three categories which are worthy of consideration. A prime example would be the scheduling problems with non-regular objective functions (Kanet & Sridharan, 2000), which have drawn much attention. Some open problems in the three fixed-job-sequence issues under study also remain to be solved in further work. In the assembly flowshop batching problem with a fixed job sequence, the max-batch model and other performance measures, such as the maximum lateness and the total tardiness could be considered. For the two-stage differentiation flowshop problem with fixed job sequences, the reverse production scheduling model could be investigated. As for the single-machine coupled-task problem subject to a fixed job sequence, determining its complexity status is very challenging.

Based upon the theoretical results of the studied problems subject to fixed job sequence(s), future works can be conducted on these prototypical problems for investigating special cases in which an optimal job sequence can be derived by analytical approaches. On the other hand, model generalization, e.g. general machine configuration, could also be considered for the studied fixed-job-sequence problems.

Bibliography

Ageev, A.A., & Baburin, A.E. (2007). Approximation algorithms for UET scheduling problems with exact delays. *Operations Research Letters*, 35(4), 533–540.

Ageev, A.A., & Kononov, A.V. (2006). Approximation algorithms for scheduling problems with exact delays. *Lecture Notes in Computer Sciences*, 4368, 1–14.

Ahmadi, J.H., Ahmadi, R.H., Dasu, S., & Tang, C.S. (1992). Batching and scheduling jobs on batch and discrete processors. *Operations Research*, 40(4), 750–763.

Ahr, D., Békési, J., Galambos, G., Oswald, M., & Reinelt, G. (2004). An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research*, 59(2), 193–203.

Albers, S., & Brucker, P. (1993). The complexity of one-machine batching problems. *Discrete Applied Mathematics*, 47(2), 87–107.

Allahverdi, A., Gupta, J.N.D., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2), 219–239.

Allahverdi, A., Ng, C.T., Cheng, T.C.E., & Kovalyov, M.Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3), 985–1032.

Baker, K.R., & Trietsch, D. (2009). *Principles of Sequencing and Scheduling*. John Wiley & Sons.

Banderier, C., & Schwer, S.R. (2005). Why Delannoy's numbers?. *Journal of Statistical Planning and Inference*, 135(1), 40–54.

Baptiste, P. (2010). A note on scheduling identical coupled tasks in logarithmic time. *Discrete Applied Mathematics*, 158(5), 583–587.

Bauman, J., & Józefowska, J. (2006). Minimizing the earliness-tardiness costs on a single machine. *Computers & Operations Research*, 33(11), 3219–3230.

Békési, J., Galambos, G., Oswald, M., & Reinelt, G. (2009). Improved analysis of an algorithm for the coupled task problem with UET jobs. *Operations Research Letters*, 37(2), 93–96.

Blazewicz, J., Ecker, K., Kis, T., Potts, C.N., Tanas, M., & Whitehead, J. (2010). Scheduling of coupled tasks with unit processing times. *Journal of Scheduling*, 13(5), 453–461.

Cheng, T.C.E., Ding, Q., & Lin, B.M.T. (2004). A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152(1), 1–13.

Cheng, T.C.E., Gupta, J.N.D., & Wang, G. (2000a). A review of flowshop scheduling research with setup times. *Production and Operations Management*, 9(3), 262–282.

Cheng, T.C.E., & Kovalyov, M.Y. (1998). An exact algorithm for batching and scheduling two part types in a mixed shop: A technical note. *International Journal of Production Economics*, 55(1), 53–56.

Cheng, T.C.E., Lin, B.M.T., & Tian, Y. (2009). Scheduling of a two-stage differentiation flowshop to minimize weighted sum of machine completion times. *Computers & Operations Research*, 36(11), 3031–3040.

Cheng, T.C.E., Lin, B.M.T., & Toker, A. (2000b). Makespan minimization in the two-machine flowshop batch scheduling problem. *Naval Research Logistics*, 47(2), 128–144.

Cheng, T.C.E., & Wang, G. (1998). Batching and scheduling to minimize the makespan in the two-machine flowshop. *IIE Transactions*, 30(5), 447–453.

Cheng, T.C.E., & Wang, G. (1999). Scheduling the fabrication and assembly of components in a two-machine flowshop. *IIE Transactions*, 31(2), 135–143.

Coffman, E.G., Nozari, A., & Yannakakis, M. (1989). Optimal scheduling of products with two subassemblies on a single machine. *Operations Research*, 37(3), 426–436.

Colina, E.C., & Quinino, R.C. (2005). An algorithm for insertion of idle time in the single-machine scheduling problem with convex cost functions. *Computers & Operations Research*, 32(9), 2285–2296.

Davis, J.S., & Kanet, J.J. (1993). Single-machine scheduling with early and tardy completion costs. *Naval Research Logistics*, 40(1), 85–101.

Davis, T. (2006). Catalan number. *http://www.geometer.org/mathcircles/catalan.pdf*.

Della Croce, F., & Trubian, M. (2002). Optimal idle time insertion in early-tardy parallel machines scheduling with precedence constraints. *Production Planning & Control*, 13(2), 133–142.

Garey, M.R., & Johnson, D.S. (1979). *Computers and Intractability: A Guide to the theory of NP-Completeness*. W.H. Freeman and Company: New York.

Garey, M.R., Tarjan, R.E., & Wilfong, G.T. (1988). One processor scheduling with symmetrical earliness and tardiness penalties. *Mathematics of Operations Research*, 13(2), 330–348.

Glass, C.A., Potts, C.N., & Strusevich, A.V. (2001). Scheduling batches with sequential job processing for two-machine flow and open shops. *INFORMS Journal on Computing*, 13(2), 120–137.

Hall, N.G., Laporte, G., Selvarajah, E., & Sriskandarajah, C. (2003). Scheduling and lot streaming in flowshops with no-wait in process. *Journal of Scheduling*, 6(4), 339–354.

Hendel, Y., & Sourd, F. (2007). An improved earliness-tardiness timing algorithm. *Computers & Operations Research*, 34(10), 2931–2938.

Herrmann, J.W., & Lee, C.Y. (1992). Three-machine look-ahead scheduling problems. *Research Report No. 92-93*, Department of Industrial Engineering, University of Florida, FL.

Hwang, F.J., Kovalyov, M.Y., & Lin, B.M.T. (2010a). Minimization of total completion time in flowshop scheduling subject to fixed job sequences, in: *Proceedings of 12th International Workshop on Project Management and Scheduling*, Tours, France, pp. 249–252.

Hwang, F.J., Kovalyov, M.Y., & Lin, B.M.T. (2010b). Total completion time minimization in two-machine flow shop scheduling problems with a fixed job sequence. *Manuscript in revision with Discrete Optimization.*

Hwang, F.J., & Lin, B.M.T. (2011). Coupled-task scheduling on a single machine subject to a fixed job sequence, *Computer & Industrial Engineering*, doi:10.1016/j.cie.2011.01.002.

Jackson, J.R. (1955). Scheduling a production line to minimize maximum lateness. *Research Report 43, Management Science Research Report*, University of California, LA.

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

Kanet, J.J., & Sridharan, V. (2000). Scheduling with inserted idle time: problem taxonomy and literature review. *Operations Research*, 48(1), 99–110.

Kovalyov, M.Y., Potts, C.N., & Strusevich, V.A. (2004). Batching decisions for assembly production systems. *European Journal of Operational Research*, 157(3), 620–642.

Kyparisis, G.J., & Koulamas, C. (2000). Flow shop and open shop scheduling with a critical machine and two operations per job. *European Journal of Operational Research*, 127(1), 120–125.

Lee, C.Y., Cheng, T.C.E., & Lin, B.M.T. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, 39(5), 616–625.

Lee, C.Y., Uzsoy, R., & Martin-Vega, L.A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4), 764–775.

Li, H., & Zhao, H. (2007). Scheduling coupled-tasks on a single machine, in: *Proceedings of 2007 IEEE Symposium on Computational Intelligence in Scheduling*, Honolulu, Hawaii, pp. 137–142.

Lin, B.M.T., & Cheng, T.C.E. (2002). Fabrication and assembly scheduling in a two-machine flowshop. *IIE Transactions*, 34(11), 1015–1020.

Lin, B.M.T., & Cheng, T.C.E. (2005). Two-machine flowshop batching and scheduling. *Annals of Operations Research*, 113(2), 149–161.

Lin, B.M.T., & Cheng, T.C.E. (2006). Two-machine flowshop scheduling with conditional deteriorating second operations. *International Transactions in Operational Research*, 13(2), 91–98.

Lin, B.M.T., & Cheng, T.C.E. (2010). Scheduling with centralized and decentralized batching policies in concurrent open shops. *Naval Research Logistics*, doi: 10.1002/nav.20437.

Lin, B.M.T., Cheng, T.C.E., & Chou, A.S.C. (2007). Scheduling in an assembly-type production chain with batch transfer. *Omega*, 35(2), 143–151.

Lin, B.M.T., & Hwang, F.J. (2011). Total completion time minimization in a 2-stage differentiation flowshop with fixed sequences per job type. *Information Processing Letters*, 111(5), 208–212.

Mosheiov, G., & Sarig, A. (2010). Minimum weighted number of tardy jobs on an m-machine flowshop with a critical machine. *European Journal of Operational Research*, 201(2), 404–408.

Mosheiov, G., & Yovel, U. (2004). Comments on "Flow shop and open shop scheduling with a critical machine and two operations per job". *European Journal of Operational Research*, 157(1), 257–261.

Ng, C.T., & Kovalyov, M.Y. (2007). Batching and scheduling in a multi-machine flow shop. *Journal of Scheduling*, 10(6), 353–364.

Orman, A.J., & Potts, C.N. (1997). On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(2), 141–154.

Orman, A.J., Shahani, A.K., & Moore, A.R. (1998). Modelling for the control of a complex radar system. *Computers & Operations Research*, 25(3), 239–249.

Pan, Y. & Shi, L. (2005). Dual constrained single machine sequencing to minimize total weighted completion time. *IEEE Transactions on Automation Science and Engineering*, 2(4), 344–357.

Portougal V. (1997). Production scheduling in the snack-food industry. *Interfaces*, 27(6), 51–64.

Potts, C.N., & Kovalyov, M.Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120(2), 228–249.

Potts, C.N., Strusevich, V.A., & Tautenhahn, T. (2001). Scheduling batches with simultaneous job processing for two-machine shop problems. *Journal of Scheduling*, 4(1), 25–51.

Shafransky, Y.M., & Strusevich, V.A. (1998). The open shop scheduling problem with a given sequence of jobs on one machine. *Naval Research Logistics*, 45(7), 705–731.

Shapiro, R.D. (1980). Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27(3), 489–498.

Sherali, H.D., & Smith, J.C. (2005). Interleaving two-phased jobs on a single machine. *Discrete Optimization*, 2(4), 348–361.

Sourd, F. (2005). Optimal timing of a sequence of tasks with general completion costs. *European Journal of Operational Research*, 165(1), 82–96.

Szwarc, W., & Mukhopadhyay, S.K. (1995). Optimal timing schedules in earliness-tardiness single machine sequencing. *Naval Research Logistics*, 42(7), 1109–1114.

Yu, W., Hoogeveen, H., & Lenstra, J.K. (2004). Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5), 333–348.

Zhang, Y., Zhou, Z., & Liu, J. (2010). The production scheduling problem in a multi-page invoice printing system. *Computers & Operations Research*, 37(10), 1814–1821.