ABSTRACT

ORDER SCHEDULING IN DEDICATED AND FLEXIBLE MACHINE ENVIRONMENTS

by
Haibing Li

Order scheduling models are relatively new in the field of scheduling. Consider a facility with m parallel machines that can process k different products (job types). Each machine can process a given subset of different product types. There are n orders from n different clients. Each order requests specific quantities of the various different products that can be produced concurrently on their given subsets of machines; it may have a release date, a weight and a due date. Preemptions may be allowed. An order can not be shipped until the processing of all the products for the order has been completed. Thus, the finish time of an order is the time when the last job of the order has been completed.

Even though the idea is somewhat new that order scheduling measures the overall completion time of a set of jobs (i.e., an order requesting different product types) instead of the individual completion time of each product type for any given order, many applications require that decision-makers consider orders rather than the individual product types in orders.

Research into order scheduling models is motivated by their various real-life applications in manufacturing systems, equipment maintenance, computing systems, and other industrial contexts, where the components of each order can be processed concurrently on the parallel machines.

In this research, two cases of order scheduling models are studied, namely, the fully dedicated environment in which each machine can produce one and only one product type, and the fully flexible machine environment in which each machine can produce all product types. With different side constraints and objective functions, the two cases include a lot of problems that are of interest.

Special interest is focused on the minimization of the total weighted completion time, the number of late orders, the maximum lateness, and so on. On the one hand, polynomial time algorithms are proposed for some problems. One the other hand, for problems that are NP-hard, complexity proofs are shown and heuristics with their worst-case performance and empirical analyses are also presented.

ORDER SCHEDULING IN DEDICATED AND FLEXIBLE MACHINE ENVIRONMENTS

by
Haibing Li

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2005

APPROVAL PAGE

ORDER SCHEDULING IN DEDICATED AND FLEXIBLE MACHINE ENVIRONMENTS

Haibing Li

Dr. Joseph Y-T. Leung, Dissertation Advisor — Date
Distinguished Professor of Computer Science, New Jersey Institute of Technology

Dr. Michael L. Pinedo, Committee Member — Date
Julius Schlesinger Professor of Operations Management, New York University

Dr. Alexandros V. Gerbessiotis, Committee Member — Date
Associate Professor of Computer Science, New Jersey Institute of Technology

Dr. Chengjun Liu, Committee Member — Date
Assistant Professor of Computer Science, New Jersey Institute of Technology

Dr. Marvin K. Nakayama, Committee Member — Date
Associate Professor of Computer Science, New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author:	Haibing Li
Degree:	Doctor of Philosophy
Date:	May 2005

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2005
- Master of Science in Computer Science,
  National University of Singapore, Singapore, 2002
- Bachelor of Engineering in Telecommunications,
  Chongqing University of Posts & Telecommunications, Chongqing, P. R. China, 1995

Major: Computer Science

Presentations and Publications:

J. Y-T. Leung, H. Li, M. L. Pinedo, and J. Zhang, "Minimizing Total Weighted Completion Time when Scheduling Orders a Flexible Environment with Uniform Machines," submitted.

J. Y-T. Leung, H. Li, and M. L. Pinedo, "Approximation Algorithms for Minimizing Total Weighted Completion Time of Orders on Parallel Identical Machines," submitted.

J. Y-T. Leung, H. Li, and M. L. Pinedo, "Scheduling Orders for Multiple Product Types to Minimize Total Weighted Completion Time," submitted.

J. Y-T. Leung, H. Li, and M. L. Pinedo, "Order Scheduling in an Environment with Dedicated Resources in Parallel," *Journal of Scheduling*, accepted for publication.

J. Y-T. Leung, H. Li, and M. L. Pinedo, "Scheduling Orders for Multiple Product Types with Due Date Related Objectives," *European Journal of Operational Research*, accepted for publication.

J. Y-T. Leung, H. Li, M. Pinedo, and C. Sriskandarajah, "Open Shops with Jobs Overlap - Revisited," *European Journal of Operational Research*, vol. 163, pp. 569–571, 2005.

H. Li and A. Lim, "Local Search with Annealing-like Restarts to Solve the VRPTW," *European Journal of Operational Research*, vol. 150, pp. 115–127, 2003. Also in *Proceedings of the 17th ACM Symposium on Applied Computing*, Madrid, Spain, pp. 560–567, 2002.

B. Cheang, H. Li, A. Lim, and B. Rodrigues, "Nurse Rostering Problems: A Bibliographic Survey," *European Journal of Operational Research*, vol. 151, pp. 447–460, 2003.

H. Li and A. Lim, "A Metaheuristic for the Pickup and Delivery Problem with Time Windows," *International Journal on Artificial Intelligent Tools*, vol. 12, pp. 173–186, 2003. Also in *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence*, pp. 151–158, 2001.

J. Y-T. Leung, H. Li, and M. L. Pinedo, "Order Scheduling Models with Applications in Practice," *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, Nottingham, United Kingdom, pp. 1–9, 2003.

H. Li and A. Lim, "A Hybrid AI Approach for Nurse Rostering," *Proceedings of the 18th ACM Symposium on Applied Computing*, Melbourne, Florida, U.S.A, pp. 730–735, 2003.

H. Li, A. Lim, and H. Lim, "Solve the Pickup and/or Delivery Problem," *The Pacific Asia Conference on Information Systems 2002*, Tokyo, Japan, 2002.

To the Lord, for His mercy, grace, and blessings.

To my beloved family and Pei Liu, for their constant love and support.

Haibing Li

ACKNOWLEDGMENT

I sincerely thank my advisor Dr. Joseph Leung. He accepted me as his student when I was unsure of theoretical work and introduced me to participate in an NSF-funded project for my dissertation research. During the past three and half years, he has spent time with me discussing every detail of my research and has provided very much insightful advice and many helpful directions. I was really lucky to work with him from the very beginning. His wonderful lectures in Computational Complexity and Scheduling Theory directly prepared a solid foundation for my dissertation research. I learned very much from him in doing research with preciseness and carefulness. His persistence, confidence, experience, and intuition in tackling problems, together with his great teaching has always been an inspiration.

I also acknowledge my sincere thanks to Dr. Michael Pinedo in New York University. He is a second advisor to me. In doing the collaborative research project, I learned, from the brainstorming discussions with him and Dr. Leung, to think and tackle problems from different angles. Dr. Pinedo's constructive comments in the discussions broadened my insight on solving problems and were very helpful for improving the results. He helped me to shape my writing and improve my communication skills. I am grateful for the lunches he provided after group meetings in NYU, the summer jobs he offered and paid me for, his many travels to NJIT to meet with us, and his enthusiasm for introducing me to industries.

Dr. Alexandros Gerbessiotis' classes taught me many ideas and techniques on the design and analysis of algorithms. His challenging problem sets gave me many exercises on solving problems with good approaches. In my research, I benefitted very much from the solid foundation built in his classes. In addition, the PC cluster environment he provided helped me on some simulation work. Thanks to him also for serving on my committee.

Many thanks to Dr. Chengjun Liu and Dr. Marvin Nakayama for being my committee members. In his classes, Dr. Liu introduced me to the field of Pattern Recognition which may become one of my areas of interest in the future. Dr. Nakayama provided constructive suggestions on my simulation work. Thanks to Dr. Jiawei Zhang for his helpful discussions of the most recent paper that I have done. Thanks to Dr. Jian Yang for his help on ILOG CPLEX. Thanks to Dr. Artur Czumaj for his lectures given in the Graph Theory classes. Thanks also to all other professors in the department who gave me lectures in various courses.

I greatly acknowledge my thanks to New Jersey Institute of Technology and the Department of Computer Science. The two institutions have provided a wonderful environment and all kinds of necessary support for my study.

The research project of order scheduling is supported by the National Science Foundation through grant DMI-0300156 to Dr. Joseph Leung and grant DMI-0245603 to Dr. Michael Pinedo.

With many friends, I have spent years of happy life in NJIT and off campus. Special thanks to Joseph Tsai, Wendy Tsai, James Hong, Peixin Lin, Shouxian Cheng, Nancy Liao, Wugang Xu, Lin Zhou, Wen Chen, Kang Wu, Yinghui Cai, Yingwei Lai, Qiong Shen, Jianghui Liu, Wenxin Mao, Paul Luor, Lee Tung, Luke Chen, Kaiyuan Chin, Tammy Yang, Rencun Huang, Dina Ding, Wantin Ong, Yang Li, Wen He, Fang Liu, Sai Li, and many other friends, for their sharing and comfort in spirits. I also thank Hairong Zhao, Yumei Huo, Xin Wang, and Chang Liu, with whom I shared the lab and discussed many study issues.

Finally, it will never be enough for me to thank my family, Pei Liu and her family, for their unconditional love, comfort, and encouragement.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
1.1 The Order Scheduling Models	1
1.2 Application Examples	4
1.3 Notation	6
1.4 Literature Survey	7
1.5 Organization and Overview of the Dissertation	10
1.5.1 The Fully Dedicated Case – $\sum C_j$	10
1.5.2 The Fully Dedicated Case – Due Date Related Objectives	10
1.5.3 The Fully Dedicated Case – $\sum w_j C_j$	11
1.5.4 The Fully Flexible Case with Identical Machines – $\sum w_j C_j$	12
1.5.5 The Fully Flexible Case with Uniform Machines – $\sum w_j C_j$	12
2 THE FULLY DEDICATED CASE – THE TOTAL COMPLETION TIME	14
2.1 Introduction	14
2.2 Preliminary Results	15
2.3 Complexity Result	19
2.4 Simple Heuristics and Their Worst-Case Analyses	24
2.5 Empirical Analysis	29
2.5.1 Schedule Generation by Heuristics	29
2.5.2 Postprocessing with Tabu Search	30
2.5.3 Generation of Problem Instances	33
2.5.4 Experimental Results and Analyses	33
2.6 Concluding Remarks	35
3 THE FULLY DEDICATED CASE – DUE DATE RELATED OBJECTIVES	38
3.1 Introduction	38

TABLE OF CONTENTS
(Continued)

Chapter Page

3.2 Min-Max Objectives 39

3.3 Minimizing the Total Number of Late Orders $\sum U_j$ 40

3.3.1 A Greedy Algorithm for the Common Due Date Case 44

3.3.2 A Heuristic for $PD \mid\mid \sum U_j$ 50

3.3.3 An Exact Algorithm for $PD \mid\mid \sum U_j$ 53

3.4 Empirical Analysis 58

3.5 Concluding Remarks 62

4 THE FULLY DEDICATED CASE – THE TOTAL WEIGHTED COMPLETION TIME 64

4.1 Introduction 64

4.2 Priority Rules for $PD \mid\mid \sum w_j C_j$ 65

4.3 Analyses of Priority Rules with Constraints on Processing Times . . 73

4.3.1 Additional Constraints $\sum_{i=1}^{m} p_{ij}/m \geq \delta_j$ 73

4.3.2 Additional Constraints $max_{1 \leq i \leq m}\{p_{ij}\} \leq 3min_{1 \leq i \leq m}\{p_{ij}\}$ . . . 78

4.4 LP-Based Approximation Algorithms 81

4.4.1 Algorithm Based on a Completion Time Formulation 81

4.4.2 Algorithm Based on a Time Interval Formulation 84

4.5 Empirical Analyses of the Algorithms 87

4.5.1 Generation of Problem Instances 87

4.5.2 Experimental Results and Analyses 88

4.6 Concluding Remarks 102

5 THE FULLY FLEXIBLE CASE WITH IDENTICAL MACHINES – THE TOTAL WEIGHTED COMPLETION TIME 104

5.1 Introduction 104

5.2 Heuristics 105

5.2.1 Sequential Two-Phase Heuristics 106

5.2.2 Dynamic Two-Phase Heuristics 109

TABLE OF CONTENTS
(Continued)

Chapter Page

5.3 Worst-Case Analyses of the Heuristics 111

5.4 Empirical Analyses of the Heuristics 127

5.5 Concluding Remarks 132

6 THE FULLY FLEXIBLE CASE WITH UNIFORM MACHINES - THE TOTAL WEIGHTED COMPLETION TIME 139

6.1 Introduction 139

6.2 Approximation Algorithm for the Nonpreemptive Case 142

6.3 Approximation Algorithm for the Preemptive Case 146

6.4 Empirical Analysis 160

6.5 Concluding Remarks 164

7 CONCLUSIONS 165

APPENDIX A PROOF OF LEMMA 2.6 169

APPENDIX B PROOF OF LEMMA 5.6 187

REFERENCES 193

LIST OF TABLES

Table		Page
2.1	The Percentage that Each Heuristic Performs the Best	34
2.2	Comparison of Average Costs when $\kappa = 2$	36
2.3	Comparison of Average Costs when $\kappa = m$	36
2.4	Comparison of Average Costs when $\kappa = r$	37
3.1	Optimality Percentage and Average CPU Time (in Seconds) vs. Each Combination of n and m	59
3.2	Optimality Percentage vs. Each Setting of Δ_1 and Δ_2 for $n = 100, 200$	60
3.3	Optimality Percentage of the *GHM* Heuristic vs. Each Combination of n and m	61
3.4	Average Improvement per Instance Obtained by the Exact Algorithm from *GHM* for Each Combination of n and m	61
4.1	The Frequency that Each Algorithm Performs the Best for Instances of Group G_1	90
4.2	The Frequency that Each Algorithm Performs the Best for Instances of Group G_2	91
4.3	The Frequency that Each Algorithm Performs the Best for Instances of Group G_3	92
4.4	The Frequency that Each Algorithm Performs the Best for Instances of Group G_4	93
4.5	The Frequency that Each Algorithm Performs the Best for Instances of Group G_5	94
4.6	Comparison of Average Costs in Percentage and Average Running Times for Group G_1	95
4.7	Comparison of Average Costs in Percentage and Average Running Times for Group G_2	96
4.8	Comparison of Average Costs in Percentage and Average Running Times for Group G_3	97
4.9	Comparison of Average Costs in Percentage and Average Running Times for Group G_4	99

LIST OF TABLES
(Continued)

Table **Page**

4.10 Comparison of Average Costs in Percentage and Average Running Times for Group G_5 100

4.11 The Percentage that the Average Costs of LP_2^2 and LP_2^3 Are Larger Than That of LP_2^1 101

5.1 The Time Complexities for the Sequential Two-Phase Heuristics 110

5.2 The Frequency and Percentage that Each Heuristic Performs the Best: $k = 2, 5, 10$ 131

5.3 The Frequency and Percentage that Each Heuristic Performs the Best: $k = 20, 50, 100$ 131

5.4 Comparison of Average Costs in Percentage: $k = 2$ 132

5.5 Comparison of Average Costs in Percentage: $k = 5$ 133

5.6 Comparison of Average Costs in Percentage: $k = 10$ 134

5.7 Comparison of Average Costs in Percentage: $k = 20$ 135

5.8 Comparison of Average Costs in Percentage: $k = 50$ 136

5.9 Comparison of Average Costs in Percentage: $k = 100$ 137

5.10 Comparison of Average Running Times over the 10 Problem Instances of $n = 200, m = 20$ and $k = 50$ 137

6.1 The Average Ratio of H_{NP} and H_P over the Instances Generated for Each Combination of n, m, and k 163

7.1 The Complexity Status of the Order Scheduling Problems Studied in the Dissertation and in the Literature 166

7.2 m-Approximation Algorithms in the Dissertation 167

LIST OF FIGURES

Figure		Page
2.1	Illustrating the proof of Lemma 2.2.	17
2.2	Neighborhood generating operators.	32
5.1	A bad example for each heuristic $H \in \{$ *WSLM-LPT*, *WSMM-MF*, *WECT-LPT*, *WECT-BIN*, *WECT-MF* $\}$.	117
5.2	Illustrating the proof of Lemma 5.7.	119
6.1	The profile after l^* is assigned	144
6.2	The schedule produced by H_{NP}.	146
6.3	Illustrating the assignment in Step 2 of procedure *Interval-Preemptive-Schedule*.	149
6.4	The schedule produced by H_P.	154
6.5	Some subcases in Case 3.	156
6.6	The subcases in Case 4.	158

CHAPTER 1

INTRODUCTION

1.1 The Order Scheduling Models

Scheduling involves allocating scarce resources to tasks over time, with the goal of optimizing one or more objectives. It plays an important role in helping decision-makers to plan good schedules for manufacturing and production systems as well as computer systems. In the last several decades, very much work has been dedicated to the field of scheduling theory, which includes various set of models, complexity results, and algorithms. For excellent books and surveys on scheduling, the reader is referred to [5, 27, 39, 43, 57]. For general topics about data structures and algorithms, the reader is referred to [12, 61]. The reader is also referred to [19, 56] for good introduction to computational complexity. Finally, the reader is referred to [30, 64] for various topics about approximation algorithms.

This dissertation focuses on a class of relatively new scheduling models, namely, ***Order Scheduling***. In what follows, the order scheduling models are formally described.

Consider a facility with m parallel machines that can produce k different product types. Assume there are n orders from n different clients. Each order $j = 1, 2, \ldots, n$ requests a quantity of product type (job) $l = 1, 2, \ldots, k$ which requires $p_{lj} \geq 0$ units of processing. If $p_{lj} = 0$, it implies that order j does not request product type l. Each order j may also have a *release date* r_j which denotes the time the order arrives in, a *due date* d_j which represents the committed shipping date, and a weight w_j which ranks the importance of order j. Note that the product types of order j are allowed to be completed after its due date, but then a penalty is incurred. When a due date *must* be met, it is referred to as a *deadline* and denoted by $\bar{d}_j$. In certain environments, there may exist *precedence* constraints that require one or more orders be completed before another order is allowed to start its processing.

For any order j, each product type $l = 1, 2, \ldots, k$ can be produced on a subset of the m machines, namely $M_l \subseteq \{1, 2, \ldots, m\}$. The m machines can be *identical*, *uniform*, or *unrelated*. In an identical machine environment, a product type l of order j requires $p_{lij} = p_{lj}$ units of processing time on any machine $i \in M_l$ if each machine is assumed to have a speed of 1. In a uniform machine environment, each machine i has a speed $v_i > 0$, and thus a product type l of order j, if entirely processed on a machine $i \in M_l$, would take $p_{lij} = p_{lj}/v_i$ units of time to process. Finally, in an unrelated machine environment, the machines have different capabilities and performance on processing a product type l. In other words, the speed of a machine $i \in M_l$ on product type l, v_{li}, is dependent on both the machine and product type; product type l of order j requires $p_{lij} = p_{lj}/v_{li}$ units of processing time on machine i.

The parallel machine environment can be either *nonpreemptive* or *preemptive*. In a nonpreemptive environment, each product type of an order must be processed in an uninterrupted fashion on a machine. In contrast, in some machine environments, preemptions are allowed, that is, a product type of an order can be processed for a period of time, interrupted and continued at a later point in time either on the same machine or on another machine. In the order scheduling models, it is assumed that preemptions disallow multiple machines to produce the same product for one order at the same time.

Product of the same type l but of different orders can be processed on a machine $i \in M_l$ in a *batch*, without setup time being incurred between orders. However, prior to processing the whole batch of type l, a setup time may incur. When a batch of type l is immediately preceded by a different batch of type l', the setup time on machine i is denoted by $s_{il'l}$. If l is the first product type on machine i, then the setup time is denoted by s_{i0l}. If for each l, $s_{il'l} = s_{i0l} = s_{il}$, then the setup times on machine i are *sequence independent*; otherwise, they are *sequence dependent.* If for each machine i, $s_{il'l} = s_{l'l}$ for all l' and l including the case $l' = 0$, then the setup times are *machine independent*; otherwise, they are *machine dependent.* If the setup times are both sequence independent and machine independent, then $s_{il'l} = s_{i0l} = s_{il} = s_l$ for each machine i. Furthermore, if the setup times are sequence independent, machine independent and *product type independent*, then $s_{il'l} = s_{i0l} = s_{il} = s_l = s$ for each machine i. When $s = 0$, no setup time is considered.

The objectives to be optimized are always a function of the completion time of the orders, which are dependent on the schedule. The *completion time* of the order j, denoted by C_j, is the time when the latest product type of this order is finished on one of the machines. Let C_{lj} denote the individual completion time of product type l of order j on one machine, it is obvious that

$$C_j = \max_l \{C_{lj}\}.$$

The objectives may also be functions of the due dates. The *lateness* of order j is defined as

$$L_j = C_j - d_j.$$

Note that it is positive when order j is completed later than its due date and negative when it is completed early. The *tardiness* of order j is defined as

$$T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}.$$

Thus, tardiness is always non-negative. The *unit penalty* of order j is defined as

$$U_j = \begin{cases} 1, & \text{if } C_j > d_j; \\ 0, & \text{otherwise.} \end{cases}$$

Several objectives are of interest, namely, the *makespan* $C_{\max} \equiv \max_{1 \le j \le n}\{C_j\}$, the *maximum lateness* $L_{\max} \equiv \max_{1 \le j \le n}\{L_j\}$, the *total weighted completion time* $\sum w_j C_j$, the *total weighted tardiness* $\sum w_j T_j$, and the *total weighted number of late orders* $\sum w_j U_j$.

The class of models described above are very rich. The two extreme cases that are of interest are:

(i) The *fully dedicated* case: There are m machines and m product types; each machine $i = 1, 2, \ldots, m$ can produce only one type of the m product types. In other words, there exists a one-to-one mapping between the machine set and the product set. This implies that $k = m$; for each $l = 1, 2, \ldots, k$, $|M_l| = 1$ and $M_l \neq M_{l'}$ if $l \neq l'$.

(ii) The *fully flexible* case: There are m machines and each machine is capable of producing all k products, that is, for each product type $l = 1, 2, \ldots, k$, $M_l = \{1, 2, \ldots, m\}$.

Of course, there are many models in between these two extreme cases. It should be noted that, for the fully dedicated case, since each machine is dedicated to produce only one product type, the machine environment types (identical, uniform, and unrelated) do not make much sense. Therefore, for this case, machine environment types are not differentiated.

1.2 Application Examples

The research into the classes of models is strongly motivated by their many practical applications in various fields. Indeed, even though the idea of measuring the overall completion time of a set of jobs (i.e., an order requesting different product types) instead of the individual completion time of each product type for any given order is somewhat new, in many contexts there are several reasons for decision-makers to consider orders as a whole rather than the individual product types in orders. First, shipping partial orders inevitably causes additional shipping costs. Second, it also causes extra management effort. Finally, some customers may require suppliers to ship complete orders. Therefore, suppliers have to wait until all products of an order are ready.

Any Make-To-Order environment at a production facility with a number of resources in parallel gives rise to the order scheduling models. Julien and Magazine [34] presented two interesting applications of the models described. The first example involves a manufacturer of computer peripherals such as terminals, keyboards and disk drives. Small businesses that purchase new computer systems order different quantities of each of these peripherals, and often require their entire order to be shipped together. From the manufacturer's point of view, it is advantageous to aggregate the demand of each peripheral and produce large batches in order to minimize the number of setups.

A second example that illustrates the models described is a pharmaceutical company that can produce different types of pills. Each type of pill needs to be bottled separately. However, for a given pill type, it is often necessary to have different bottle sizes. The pills and the bottles may be produced based on forecasts. However, the bottling and packaging stage is order driven. The customers, which may be drugstores or hospitals, order certain quantities of each product type (a product type being a bottle of a given size of a given pill type). The production setups are the switch-overs in the bottling facility.

Yang [67] gave yet another example in the car repair shop. Suppose each car has several broken parts that need to be fixed. Each broken part can only be fixed by a certain set of mechanics in the shop. Several mechanics can work concurrently at different parts of the same car. When a mechanic finishes his work on a car, he can continue working on his assigned broken parts in another car. A car can not leave the shop until every broken part is fixed.

In manufacturing systems involving two stages, different types of components (or subassemblies) are produced first at its pre-assembly stage, and then assembled into final products (jobs) at its assembly stage. The pre-assembly stage consists of parallel machines (called feeding machines), each of which produces its own subset of components. Each assembly operation can not start its processing until all the necessary components are fed in. As illustrated in [15, 40, 58], there are a lot of application examples of such two-stage assembly systems. A good example arises in parallel computing systems, in which several programs (or tasks) are independently processed first at certain processors, and then gathered at a main processor for final data-processing. The main processor can only start its processing after all the programs feed in their results. As noted by Sung and Yoon [63], the order scheduling models only deal with the pre-assembly stage in such two-stage systems.

1.3 Notation

For convenience of description in later chapters, the following notation is proposed for the class of order scheduling problems that are of interest. The notation is an extension of the $\alpha \mid \beta \mid \gamma$ notation introduced by Graham, Lawler, Lenstra, and Rinnooy Kan [27].

In the α field, the machine environment of the fully dedicated case is denoted by PDm, where the m denotes the number of machines; when the m is omitted it is assumed that the number of machines is arbitrary. For the fully flexible case, the *identical*, *uniform*, and *unrelated* parallel machine environment are denoted by PFm, QFm, and RFm, respectively; again, when the m is omitted it implies that the number of machines is arbitrary.

In the β field, Πk is included to refer to the fact that there are k different product types; the absence of the k indicates that the number of different product types is arbitrary. In addition, an s, s_l, or s_{li} may be included in the β field to indicate different types of setup times that have been defined previously. Note that setup times do not make sense for the fully dedicated case, since each machine is dedicated to produce a unique product type. Other constraints that can be included in the β field are release dates r_j, preemptions ($prmp$), precedence constraints ($prec$), processing time properties, and so on.

The γ field denotes the objective function to be optimized, for example, the maximum lateness L_{max} and the total weighted number of late orders $\sum w_jU_j$.

As an example of the notation, $PDm \mid r_j \mid \sum T_j$ refers to the case with m fully dedicated machines in parallel, n different orders with order j having a release date r_j and a due date d_j. The objective is the minimization of the total tardiness. As another example, $PF \mid r_j, \Pi, prmp \mid \sum w_jC_j$ refers to a fully flexible environment with an arbitrary number of identical machines in parallel. The number of different product types is arbitrary. Order j has a release date r_j, and preemptions are allowed. The objective is the minimization of the total weighted completion time. Finally, the notation $QF6 \mid prmp, s, \Pi3, d_j = d \mid L_{max}$ refers to the fully flexible case with 6 uniform machines in parallel and 3 different product types. Each product type has the same setup time s, order j has a common due date d, and preemptions are allowed. The objective is the minimization of maximum lateness.

1.4 Literature Survey

For the order scheduling models, most of the past work has focused on the fully dedicated case.

Sung and Yoon [63] considered a special case where $m = 2$ and all $r_j = 0$, namely, $PD2 \mid\mid \sum w_jC_j$. They showed that $PD2 \mid\mid \sum w_jC_j$ is strongly NP-hard. For the special case with all identical weights, Wagneur and Sriskandarajah [65] tried to prove that $PD2 \mid\mid \sum C_j$ is strongly NP-hard. However, Leung et al. [49] recently pointed out that their proof is not correct. Wang and Cheng [66] proposed three greedy heuristics for $PD \mid\mid \sum w_jC_j$. Two of the heuristics generalized the two greedy heuristics designed for $PD2 \mid\mid \sum w_jC_j$ by Sung and Yoon [63]. All these three heuristics have an approximation ratio of m. Note that a *ρ-approximation algorithm* is a polynomial-time algorithm that produces a solution of objective cost at most ρ times the optimal cost. Wang and Cheng [66] also proposed a $\frac{16}{3}$-approximation algorithm for $PD \mid\mid \sum w_j C_j$. The algorithm is based on a linear programming relaxation which is formulated on the time intervals geometrically divided over the time horizon.

For due date related objectives, Wagneur and Sriskandarajah [65] showed that $PD2 \mid\mid \sum U_j$ is NP-hard in the ordinary sense. Cheng and Wang [10] showed that there exists a pseudo-polynomial time algorithm for every fixed $m \geq 2$. Ng, Cheng, and Yuan [55] showed that when the number of machines is arbitrary the problem becomes strongly NP-hard; they showed that the more restricted problem $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$ is strongly NP-hard as well. For $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$, Ng, Cheng, and Yuan [55] proposed a d-approximation heuristic based on a linear programming relaxation.

It seems that the fully flexible case is more complicated. However, there is still some work done for it. When there is no setup time for each product type, Yang [67] showed that the problem $PF2 \mid \Pi \mid \sum C_j$ is NP-hard in the ordinary sense even if each order requests only one or two product types. However, it is not known whether or not there exists a pseudo-polynomial time algorithm for the above ordinary NP-hard problem. When the number of machines is arbitrary, Blocher and Chhajed [4] showed that the problem $PF \mid \Pi \mid \sum C_j$ is NP-hard in the strong sense. Blocher and Chhajed [4] also developed five heuristics for solving the problem and reported some experimental results.

When preemptions are allowed, Leung et al. [44] showed that the problem $PF2 \mid prmp, \Pi2 \mid \sum C_j$ is NP-hard in the ordinary sense. They also developed a $(2 - 1/m)$-approximation algorithm for a more general problem $PF \mid prmp, \Pi \mid \sum w_j C_j$.

Introducing setup times for the product types usually makes the problems much harder, even for a single machine. Gerodimos et al. [20] studied $PF1 \mid s_l, \Pi k \mid \gamma$ and $PF1 \mid s_l, \Pi \mid \gamma$. They showed that $PF1 \mid s_l, \Pi \mid L_{max}$ is NP-hard in the ordinary sense, and it can be solved by dynamic programming in $O(k^2 n^k)$, which is polynomial for fixed k. If it requires all operations for any product type be scheduled contiguously, i.e., the operations for each product type must be scheduled in one batch, such constraints are termed as *group technology* (GT) in [20, 54]. With group technology, Gerodimos et al. [20] showed that $PF1 \mid s_l, \Pi, GT \mid L_{max}$ can be solved in $O(nk + (n+k)\lg(n+k))$ time.

For the objective of minimizing the total weighted number of tardy orders, Gerodimos et al. [20] showed that $PF1 \mid s, \Pi 2 \mid \sum U_j$ is ordinary NP-hard, while $PF1 \mid s = 1, \Pi, p_{lj} \in \{0, 1\} \mid \sum U_j$ is strongly NP-hard. They gave an $O(nk^2 d_{max}^{k+1})$ time algorithm for the more general problem $PF1 \mid s_l, \Pi \mid \sum w_j U_j$, where $d_{max} = \max_j\{d_j\}$. Thus, the algorithm is pseudo-polynomial for $PF1 \mid s_l, \Pi k \mid \sum U_j$. Recently, Cheng, Ng, Yuan [9] showed a stronger result that even the very restricted problem $PF1 \mid s = 1, \Pi, d_j = d, p_{lj} \in \{0, 1\}, \sum_l p_{lj} = p \mid \sum U_j$ is strongly NP-hard. Interestingly, they noticed that $PF1 \mid s_l, \Pi, d_j = d, p_{lj} > 0 \mid \sum U_j$ can be solved by the shortest processing time (SPT) rule.

For the objective of minimizing the total completion time, Gerodimos et al. [20] showed that $PF1 \mid s_l, \Pi, GT, p_{lj} > 0 \mid \sum C_j$ can be solve in $O(kn \lg n)$ time. Ng, Cheng, and Yuan [54] showed that both $PF1 \mid s, \Pi, GT, p_{lj} \in \{0, 1\} \mid \sum C_j$ and $PF1 \mid s, \Pi, p_{lj} \in \{0, 1\} \mid \sum C_j$ are strongly NP-hard. Interestingly, Ng, Cheng, and Yuan [54] posed that the complexity of $PF1 \mid s_l, \Pi, p_{lj} > 0 \mid \sum C_j$ remains an open problem. Note that the first problem is not necessarily a special case of the latter one. When k is fixed, the complexity of $PF1 \mid s_l, \Pi k, GT \mid \sum C_j$ and $PF1 \mid s_l, \Pi k, \mid \sum C_j$ remains open (see Gerodimos et al. [20]).

1.5 Organization and Overview of the Dissertation

In this dissertation, special attention is focused on two aspects: i) to establish computational complexity results for some problems; ii) to design algorithms for some NP-hard problems, and analyze the algorithms both theoretically and empirically.

The dissertation is based on the results contained in [48, 46, 47, 45, 50]. In what follows, the organization and overview of the dissertation will be described.

1.5.1 The Fully Dedicated Case – $\sum C_j$

Chapter 2 presents some results obtained for the fully dedicated order scheduling model, with the objective of minimizing the total (unweighted) completion time $\sum C_j$. First of all, some structural properties are shown for a class of problems. Then, the problem $PD3 \mid\mid \sum C_j$ is shown to be NP-hard in the strong sense. Thus, it is unlikely that the problem can be solved by any polynomial-time algorithms unless $\mathcal{P} = \mathcal{NP}$. Due to this, two new heuristics together with their worst-case performance analyses are presented. In addition, the chapter also presents an empirical analysis of these two new heuristics together with the three other heuristics proposed by Sung and Yoon [63] and by Wang and Cheng [66]. For the purpose of comparison, a Tabu Search procedure is implemented to improve the best schedule generated by the five heuristics. The results show that one of two heuristics proposed in this chapter performs the best among all five algorithms.

1.5.2 The Fully Dedicated Case – Due Date Related Objectives

Chapter 3 focuses on the fully dedicated case with some due date related (unweighted) objectives. It shows that the *EDD* (Earliest Due Date first) rule (see Jackson [32]) solves $PD \mid\mid L_{\max}$; the preemptive *EDD* rule solves the problem $PD \mid r_j, prmt \mid L_{\max}$; and $PD \mid prec \mid L_{\max}$ can be solved by dynamic programming. In addition, the problem $PD \mid \bar{d}_j \mid L_{\max}$ can be solved in polynomial time. For a special case with common due dates, namely $PD \mid d_j = d \mid \sum U_j$, the chapter gives a greedy heuristic which has a worst-case performance ratio of

$$\mathcal{H}\left(\max_{1\leq j\leq n}\left\{\sum_{i=1}^{m}\frac{p_{ij}}{c_i}\right\}\right),$$

where $\mathcal{H}(k) \equiv \sum_{i=1}^{k}\frac{1}{i}$ is the harmonic series (see Cormen, Leiserson, Rivest, and Stein [12]), and $c_i = GCD(p_{i1}, p_{i2}, \ldots, p_{in}, d)$, for $i = 1, 2, \ldots, m$. Furthermore, the Hodgson-Moore algorithm (see Moore [51]) is generalized for solving $PD \mid\mid \sum U_j$. Based on the structural properties of the problem $PD \mid\mid \sum U_j$, an exact algorithm, which is based on constraint propagation, backtracking, and bounding techniques, is designed. Finally, an empirical analysis is presented for the two algorithms that are proposed for $PD \mid\mid \sum U_j$.

1.5.3 The Fully Dedicated Case – $\sum w_j C_j$

Chapter 4 still focuses on the fully dedicated case. However, the objective of interest is the minimization of the total weighted completion time $\sum w_j C_j$. Special interest is focused on the design and analysis of some approximation algorithms for $PD \mid\mid \sum w_j C_j$ and $PD \mid r_j \mid \sum w_j C_j$, both of which are NP-hard in the strong sense.

The algorithms considered are of two types: priority rules (either static or dynamic) and LP-based algorithms. The priority rules, generalized from the heuristics for $PD \mid\mid \sum C_j$ in Chapter 2, are only applicable to $PD \mid\mid \sum w_j C_j$, i.e., when $r_j = 0$ for all j. Even though the worst-case bounds of the priority rules indicate that they could perform badly, some small constant bounds (2 or 3) can be obtained for the rules when the processing times are subject to additional constraints.

The two LP-based algorithms are based on two different linear programming relaxations. Analyses show that both algorithms have constant performance ratios for solving $PD \mid r_j \mid \sum w_j C_j$, thus $PD \mid\mid \sum w_j C_j$.

The chapter also presents empirical comparisons of the various algorithms, based on their results obtained for solving a large number of instances generated for the problem $PD \mid\mid \sum w_j C_j$. The conclusions from the empirical analyses provide insights into the trade-offs with regard to solution quality, speed, memory space, and implementation complexity.

1.5.4 The Fully Flexible Case with Identical Machines – $\sum w_j C_j$

Chapter 5 examines the fully flexible case with identical machines in parallel. The objective of interest is the minimization of the total weighted completion time. According to the notation defined previously, the problem is denoted by $PF \mid \Pi k \mid \sum w_j C_j$ when k is fixed and as $PF \mid \Pi \mid \sum w_j C_j$ when k is arbitrary.

Both $PF \mid \Pi k \mid \sum w_j C_j$ and $PF \mid \Pi \mid \sum w_j C_j$ are NP-hard in the strong sense. The attention is focused on two classes of heuristics, which are referred to as sequential two-phase heuristics and dynamic two-phase heuristics. The chapter presents a worst-case analysis as well as an empirical analysis of nine heuristics. The analyses enable one to rank these heuristics according to their effectiveness, taking solution quality as well as running time into account.

1.5.5 The Fully Flexible Case with Uniform Machines – $\sum w_j C_j$

Chapter 6 investigates the fully flexible case with uniform machines in parallel. Again, the objective of interest is minimizing the total weighted completion time. Preemptions are allowed. According to the notation defined previously, when k is fixed, the non-preemptive and preemptive cases of the problem are denoted by $QF \mid \Pi k \mid \sum w_j C_j$ and $QF \mid prmp, \Pi k \mid \sum w_j C_j$, respectively. When k is arbitrary, they are denoted by $QF \mid \Pi \mid \sum w_j C_j$ and $QF \mid prmp, \Pi \mid \sum w_j C_j$, respectively.

Due to their NP-hardness, a heuristic is proposed for the non-preemptive case and the preemptive case, respectively. For the heuristic that is proposed for the nonpreemptive case, analysis shows that it has a worst-case bound of m. For the heuristic proposed for the preemptive case, analysis shows that it has a worst-case bound of m for the general case and of 2 when $m = 3$. The two heuristics are also implemented to have an empirical analysis. The observations on the experimental results reveal that the two heuristics can produce very near-optimal solutions in practice.

CHAPTER 2

THE FULLY DEDICATED CASE – THE TOTAL COMPLETION TIME

2.1 Introduction

This chapter examines the fully dedicated order scheduling model. As described in Chapter 1, in the fully dedicated case, there are m machines and m product types; each machine can produce one and only one type. Thus, these is a one-to-one mapping between the machine set and the set of product types. Without loss of generality, it may be assumed that machine i is the only machine that can produce type i and type i is the only type that can be produced on machine i. Due to this, the subscript i refers to a machine as well as to a product type. In addition, the notation p_{iij}, which denotes the time required to process product type i of order j on machine i, can be shortened as p_{ij}.

It has been noted that Wagneur and Sriskandarajah [65] referred to the fully dedicated order scheduling model as "Open shops with job overlaps", while Ng et al. [55] called this model "Concurrent open shops". This case is considerably easier than the other cases because there is no freedom in the assignment of jobs to machines. Each machine can start processing at time 0 and keeps on producing as long as there is a demand. The main issue here is the assignment of finished products to customers. For dedicated machines, the setup times do not play a role in scheduling, and can therefore be dropped from consideration.

According to the notation defined in Chapter 1, the problem $PD1 \mid \beta \mid \gamma$ is identical to the problem $1 \mid \beta \mid \gamma$. So $PD1 \mid\mid \sum T_j$, $PD1 \mid r_j \mid \sum C_j$, $PD1 \mid\mid \sum w_j U_j$ are all NP-hard (see Du and Leung [14] and Lenstra [41]). Therefore, the research is focused on $m \geq 2$.

The problem $PD \mid\mid \sum C_j$ is of special interest. The notation $PD \mid\mid \sum C_j$ implies that each order $j = 1, 2, \ldots, n$ is released at time zero, and all $w_j = 1$. For this problem, Wagneur and Sriskandarajah [65] considered the case of two machines, i.e., $PD2 \mid\mid \sum C_j$, and presented a proof claiming that the minimization of the total completion time is strongly NP-hard. Unfortunately, their proof is not correct (Leung et al., [43]). Independently, Sung and Yoon [63] showed that $PD2 \mid\mid \sum w_j C_j$ is strongly NP-hard. One of the main results in this chapter is a proof that the $PDm \mid\mid \sum C_j$ problem is strongly NP-hard for every $m \geq 3$.

Several heuristics have been proposed in the literature for the $PD \mid\mid \sum C_j$ problem. Wang and Cheng [66] analyzed three greedy heuristics whose worst-case bounds are m. Two of these heuristics were generalizations of heuristics proposed by Sung and Yoon [63] for two machines. In this chapter, two new heuristics for $PD \mid\mid \sum C_j$ are introduced. For one of these two heuristics, it is shown to have a worst-case bound of m. Experimental results show that one of the two new heuristics outperforms the three heuristics that have appeared in the literature.

The results in this chapter have been published in Leung, Li, and Pinedo [48]. The chapter is organized as follows. Section 2.2 presents some preliminary results for a fairly general class of objective functions that include the total completion time as well as the total tardiness. Section 2.3 shows that $PDm \mid\mid \sum C_j$ is NP-hard in the strong sense for every fixed $m \geq 3$. Section 2.4 gives a description and an analysis of two new heuristics. Section 2.5 presents an empirical analysis of various heuristics. Finally, some concluding remarks are presented in Section 2.6.

2.2 Preliminary Results

In this section, some preliminary results are developed for the fully dedicated order scheduling model. These results are helpful for understanding the problems and are very useful for designing algorithms for these problems. The following general properties can be shown fairly easily.

Lemma 2.1 (Structural Properties)

(i) *The makespan $C_{\max}$ is independent of the schedule, provided that the machines are always kept busy whenever there are orders available for processing (i.e., unforced idleness is not allowed).*

(ii) *If $f_j(C_j)$ is increasing in C_j for all j, then there exists an optimal schedule for the objective function $f_{\max}$ as well as an optimal schedule for the objective function $\sum f_j(C_j)$ in which all machines process the orders in the same sequence.*

(iii) *If for some machine i there exists a machine k such that $p_{ij} \leq p_{kj}$ for $j = 1, \ldots, n$, then machine i does not play any role in determining the optimal schedule and may be ignored.*

Some remarks with regard to these properties are in order. The second property does not hold for the more general problem in which the function $f_j(C_j)$ is not monotone (e.g., problems that are subject to earliness and tardiness penalties). The third property is useful for reducing the problem size.

Consider the problem $PD \mid \beta \mid \sum f_j(C_j)$. Since this problem is strongly NP-hard, it is advantageous to develop dominance conditions or elimination criteria.

Lemma 2.2 *If in the problem $PD1 \mid\mid \sum f_j(C_j)$ there are two jobs j and k such that $p_j \leq p_k$ and*

$$\frac{df_j(t)}{dt} \geq \frac{df_k(t)}{dt}$$

then there exists an optimal schedule in which job j precedes job k.

Proof: The proof is by contradiction. Suppose job j is processed after job k (see Figure 2.1). In between jobs k and j there are a number of jobs that are being processed. Assume that in this original schedule the completion of job j is denoted by C_j and the completion of job k is denoted by C_k. Consider the following interchange:

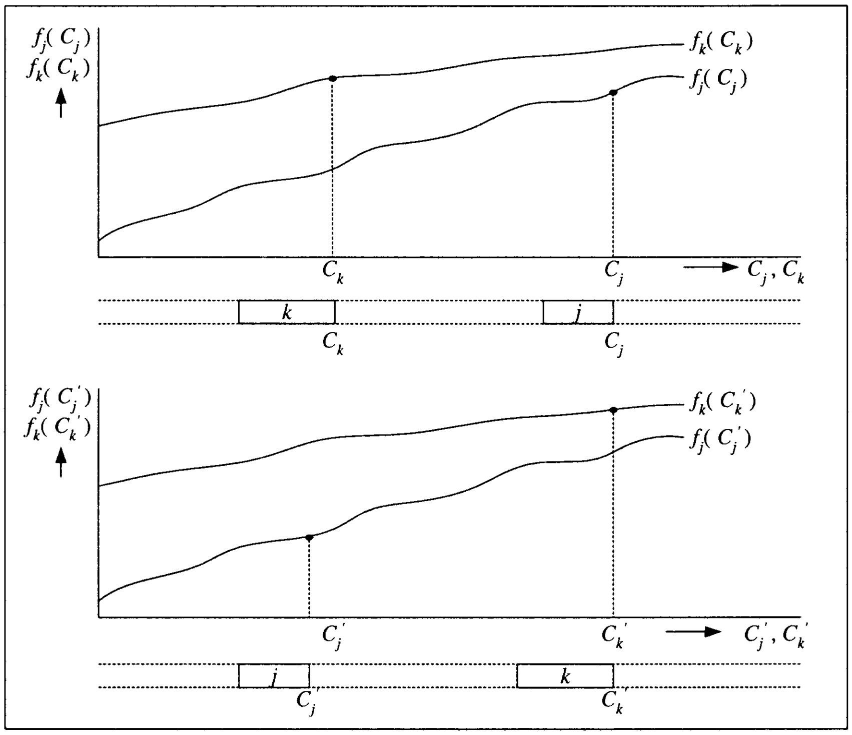


Figure 2.1 Illustrating the proof of Lemma 2.2.

Put job j in the position of job k and vice versa. Let the completion time of job j in this new schedule be denoted by C_j' and the completion of job k by C_k'. In the new schedule, all the jobs scheduled in between the two jobs are completed earlier, since job j is shorter than job k. Since the cost functions are all monotonically increasing, the contribution of these jobs to the overall objective goes down. It is clear that $C_j = C_k'$. It is also clear that $C_j' \leq C_k$. So the move of job k increases the objective function by an amount $f_k(C_k') - f_k(C_k)$, whereas the move of job j reduces the objective function by an amount $f_j(C_j) - f_j(C_j')$. Since

$$C_j - C_j' \geq C_k' - C_k$$

and

$$\frac{df_j(t)}{dt} \geq \frac{df_k(t)}{dt}$$

the savings incurred by processing job j earlier are larger than the cost incurred by processing job k later. $\square$

In a sense the result in Lemma 2.2 generalizes the well-known dominance result of Emmons [16]. Intuitively, it means that the cost function of the shorter job j has to be "steeper" than the cost function of the longer job k; the actual levels of the costs are of no importance (see Figure 2.1). A dual of the result in Lemma 2.2 can be established for the single machine problem in which the jobs are subject to earliness penalties.

Lemma 2.2 can be generalized to a result for the problem $PD \mid\mid \sum f_j(C_j)$.

Lemma 2.3 *If in the problem $PD \mid\mid \sum f_j(C_j)$ there are two orders j and k such that $p_{ij} \leq p_{ik}$ for each $i = 1, 2, \ldots, m$, and*

$$\frac{df_j(t)}{dt} \geq \frac{df_k(t)}{dt},$$

then there exists an optimal schedule in which order j precedes order k.

Consider now the special case in which $p_{ij} = p_i$ for all j. That is, the processing time requirements for all n orders on machine i are identical to p_i. For which type of objective functions is it possible to reduce the problem size by not taking machine i into consideration? It is clear that if the objective is $L_{\max}$ machine i does not play a role in determining the optimal sequence; the optimal sequence is *EDD* (which is determined by just the due dates). However, even though in this case the optimal sequence is not affected by machine i (or by any other machine for that matter), deleting machine i may affect the value of the objective function. In the case of the objective functions $f_{\max}$ and $\sum U_j$ the problem cannot be reduced in size by deleting machine i: the smaller problem without machine i may have a different optimal sequence.

2.3 Complexity Result

Wagneur and Sriskandarajah [65] tried to prove that $PD2 \mid\mid \sum C_j$ is NP-hard in the strong sense. However, Leung et al. [49] recently pointed out that their proof is not correct. In this section, the following complexity result is presented.

Theorem 2.4 *The problem $PD3 \mid\mid \sum C_j$ is strongly NP-hard.*

However, the complexity of $PD2 \mid\mid \sum C_j$ remains an open problem. In what follows, Theorem 2.4 will be proved by reducing the Numerical Matching with Target Sums (*NMTS*) problem to the decision version of $PD3 \mid\mid \sum C_j$. The *NMTS* problem is known to be strongly NP-hard (Garey and Johnson [19]) and can be stated as follows:

Definition 2.1 (NMTS) : *Let $A = \{a_1, \dots, a_n\}$, $B = \{b_1, \dots, b_n\}$, and $C = \{c_1, \dots, c_n\}$ be three sets of natural numbers. Is it possible to find a partition of $A \cup B$ into n subsets $\{a_{i(r)}, b_{j(r)}\}$ such that $c_r = a_{i(r)} + b_{j(r)}$ for $r = 1, \dots, n$?*

Given any instance of *NMTS*, one can construct an instance of the decision version of $PD3 \mid\mid \sum C_j$ with orders $J_a(J_b, J_c)$ of type a (respectively, b, or c) that have the following processing times on the 3 machines, $j = 1, 2, \dots, n$:

$J_a : (2L + a_j, 2L + X - a_j, 0)$,

$J_b : (2L + 2X + b_j, L + X - b_j, 0)$, and

$J_c : (L, 2L + 2c_j, 5L + 2X + c_j)$, where $L > nX > 2n \sum_{j=1}^{n} c_j$.

Since it is necessary that $\sum_{j=1}^{n} c_j = \sum_{j=1}^{n}(a_j + b_j)$ for any instance of *NMTS* in order to have a "Yes" answer, it is assumed that the given instance of *NMTS* satisfies this condition. Otherwise, it can be mapped into a "No" instance of the scheduling problem. The threshold for the decision version of $PD3 \mid\mid \sum C_j$ is given as:

$$D = \frac{n(15n + 7)}{2} L + (3n^2 + 2n)X + 2\sum_{j=1}^{n} c_j - \sum_{j=1}^{n} a_j + 3\sum_{j=1}^{n-1}(n - j)c_j,$$

where $c_1 \leq c_2 \leq \dots \leq c_n$.

Lemma 2.5 *If the given instance of NMTS has a solution, then there is a schedule with $\sum C_j \leq D$.*

Proof: Let $c_r = a_{i(r)} + b_{j(r)}$ for $1 \leq r \leq n$. One can construct a schedule S with the following sequence:

$$J_{a_{i(1)}}, J_{b_{j(1)}}, J_{c_1}, J_{a_{i(2)}}, J_{b_{j(2)}}, J_{c_2}, \cdots, J_{a_{i(n)}}, J_{b_{j(n)}}, J_{c_n}.$$

It is easy to show that S has $\sum C_j$ exactly D. □

The following lemma is the key to the proof of Theorem 2.4.

Lemma 2.6 *There is an optimal schedule in which the orders are scheduled in the order (abc), repeated n times.*

Proof: The proof is relegated to Appendix A. □

Lemma 2.7 *There is an optimal schedule in which all orders of type a finish on machine 2, while all orders of type b finish on machine 1.*

Proof: Let S be an optimal schedule. By Lemma 2.6, it can be assumed that the orders are scheduled in the order (abc), repeated n times. Thus, an order of type a is in position $3k+1$ and an order of type b is in position $3k+2$, $0 \leq k \leq n-1$. Let C_{3k+1} and C_{3k+2} be the finish times of the orders in positions $3k+1$ and $3k+2$, respectively. By reindexing the orders if necessary, it can be assumed that the orders in positions $3k+1$ and $3k+2$ are $J_{a_{k+1}}$ and $J_{b_{k+1}}$, respectively, $0 \leq k \leq n-1$. Then,

$$\begin{aligned} C_{3k+1} &= k(5L+2X) + 2L + \\ &\quad \max\left\{ \sum_{j=1}^{k}(a_j + b_j) + a_{k+1}, \sum_{j=1}^{k}(2c_j - a_j - b_j) + X - a_{k+1} \right\} \\ &= k(5L+2X) + 2L + X + \sum_{j=1}^{k}(2c_j - a_j - b_j) - a_{k+1}. \qquad (2.1) \end{aligned}$$

$$\begin{aligned}
C_{3k+2} &= k(5L+2X)+3L+2X+ \\
&\quad \max\left\{\sum_{j=1}^{k}(a_j+b_j)+L+a_{k+1}+b_{k+1}, \sum_{j=1}^{k}(2c_j-a_j-b_j)-a_{k+1}-b_{k+1}\right\} \\
&= k(5L+2X)+3L+2X+\sum_{j=1}^{k}(a_j+b_j)+L+a_{k+1}+b_{k+1}. \qquad (2.2)
\end{aligned}$$

(2.1) and (2.2) follow from the assumption that $L > nX > 2n\sum_{j=1}^{n} c_j$.

Thus, the order of type a in the $(3k+1)^{st}$ position finishes on machine 2 and the order of type b in the $(3k+2)^{nd}$ position finishes on machine 1. □

Lemma 2.8 *If there is a schedule with $\sum C_j \leq D$, then there is a solution to the given instance of NMTS.*

Proof: By the above two lemmas, the finish times in S can be computed as follows. (For convenience, it is assumed that S is $a_1b_1c_1a_2b_2c_2\cdots a_nb_nc_n$; otherwise, one can reindex the orders.)

$$\begin{aligned}
C_{3k+1} &= \sum_{j=1}^{k}((2L+X-a_j)+(L+X-b_j)+(2L+2c_j))+(2L+X-a_{k+1}) \\
&= k(5L+2X)+2L+X-a_{k+1}+\sum_{j=1}^{k}(2c_j-a_j-b_j). \qquad (2.3)
\end{aligned}$$

$$\begin{aligned}
C_{3k+2} &= \sum_{j=1}^{k}((2L+a_j)+(2L+2X+b_j)+L)+(2L+a_{k+1})+(2L+2X+b_{k+1}) \\
&= k(5L+2X)+4L+2X+\sum_{j=1}^{k+1}(a_j+b_j). \qquad (2.4)
\end{aligned}$$

$$\begin{aligned}
C_{3k} &= \max\left\{\begin{array}{c} \sum_{j=1}^{k}\left((2L+a_j)+(2L+2X+b_j)+L\right), \\ \sum_{j=1}^{k}\left((2L+X-a_j)+(L+X-b_j)+(2L+2c_j)\right) \\ \sum_{j=1}^{k}\left(5L+2X+c_j\right) \end{array}\right\} \\
&= k(5L+2X)+\sum_{j=1}^{k}(a_j+b_j)+\max\left\{0, 2\sum_{j=1}^{k}(c_j-a_j-b_j), \sum_{j=1}^{k}(c_j-a_j-b_j)\right\} \\
&= k(5L+2X)+\sum_{j=1}^{k}(a_j+b_j)+2\max\left\{0, \sum_{j=1}^{k}(c_j-a_j-b_j)\right\}. \qquad (2.5)
\end{aligned}$$

For ease of notation, let

$$\delta_j = c_j-(a_j+b_j), \qquad (2.6)$$

$$\lambda_k = \max\left\{\sum_{j=1}^{k}\delta_j, 0\right\}, \qquad (2.7)$$

and

$$\Sigma_a = \sum_{k=0}^{n-1} C_{3k+1},\ \Sigma_b = \sum_{k=0}^{n-1} C_{3k+2},\ \Sigma_c = \sum_{k=1}^{n} C_{3k}. \qquad (2.8)$$

Now,

$$\begin{aligned}
\Sigma_a = \sum_{k=0}^{n-1} C_{3k+1} &= \frac{5n^2-n}{2}L+n^2X-\sum_{j=1}^{n}a_j+\sum_{k=1}^{n-1}\sum_{j=1}^{k}(2c_j-a_j-b_j) \\
&= \frac{5n^2-n}{2}L+n^2X-\sum_{j=1}^{n}a_j+\sum_{j=1}^{n-1}(n-j)(c_j+\delta_j), \qquad (2.9)
\end{aligned}$$

$$\Sigma_b = \sum_{k=0}^{n-1} C_{3k+2} = \frac{5n^2+3n}{2}L+n(n+1)X+\sum_{j=1}^{n}(n+1-j)(c_j-\delta_j), \qquad (2.10)$$

and

$$\Sigma_c = \sum_{k=1}^{n} C_{3k} = \frac{5n(n+1)}{2}L+n(n+1)X+\sum_{j=1}^{n}(n+1-j)(c_j-\delta_j)+2\sum_{k=1}^{n}\lambda_k. \qquad (2.11)$$

Therefore, the total completion time in S is

$$\begin{aligned}\sum_{j=1}^{3n} C_j &= \Sigma_a + \Sigma_b + \Sigma_c \\ &= \frac{n(15n+7)}{2}L + (3n^2+2n)X + 2\sum_{j=1}^{n} c_j - \sum_{j=1}^{n} a_j + \\ &\quad 3\sum_{j=1}^{n-1}(n-j)c_j + 2\sum_{j=1}^{n}\lambda_j - \sum_{j=1}^{n}(n+2-j)\delta_j \\ &= D + 2\sum_{j=1}^{n}\lambda_j - \sum_{j=1}^{n}(n+2-j)\delta_j. \qquad (2.12)\end{aligned}$$

Since the total completion time in S is no larger than D, it follows that

$$2\sum_{j=1}^{n}\lambda_j - \sum_{j=1}^{n}(n+2-j)\delta_j \leq 0.$$

In what follows, it will be shown that the above inequality implies that $\delta_j = 0$ for all $1 \leq j \leq n$, which means that the instance of *NMTS* has a solution.

To show this, the first observation is that

$$\sum_{j=1}^{n} c_j = \sum_{j=1}^{n}(a_j+b_j) \Rightarrow \sum_{j=1}^{n}\delta_j = \sum_{j=1}^{n}((c_j-(a_j+b_j)) = 0. \qquad (2.13)$$

Now,

$$\begin{aligned}2\sum_{j=1}^{n}\lambda_j - \sum_{j=1}^{n}(n+2-j)\delta_j &= 2\sum_{j=1}^{n}\max\left\{0, \sum_{i=1}^{j}\delta_i\right\} - \sum_{j=1}^{n}(n+2-j)\delta_j \\ &= 2\max\left\{0, \delta_1, \delta_1+\delta_2, \ldots, \sum_{j=1}^{n}(n+1-j)\delta_j\right\} - \sum_{j=1}^{n}(n+2-j)\delta_j, \\ &= 2\max\left\{0, \delta_1, \delta_1+\delta_2, \ldots, \sum_{j=1}^{n}(n+1-j)\delta_j\right\} - \sum_{j=1}^{n}(n+1-j)\delta_j, \quad (2.14)\end{aligned}$$

recall that $\sum_{j=1}^{n}\delta_j = 0$.

Now, it has to be shown that

$$2\max\left\{0, \delta_1, \delta_1+\delta_2, \ldots, \sum_{j=1}^{n}(n+1-j)\delta_j\right\} \leq \sum_{j=1}^{n}(n+1-j)\delta_j \qquad (2.15)$$

implies that $\delta_j = 0$ for each $1 \leq j \leq n$.

This can be shown by contradiction. Suppose it is not the case; that is, one or more δ_j are strictly positive and one or more δ_j are strictly negative.

The claim is that it is impossible for a partial sum $\sum_{j=1}^{k} \delta_j$ to be greater than zero. If such a partial sum is positive, then the left-hand side (*LHS*) in (2.15) must be greater than 0. If the RHS in (2.15) is negative, the contradiction is immediately established; if the right-hand side (*RHS*) is positive, then the *LHS* is at least twice as large because of the last term on the *LHS* and the contradiction is also established. So a partial sum can never be strictly positive.

Consider the case where the partial sums $\sum_{j=1}^{k} \delta_j$ are always either negative or zero. If all the partial sums are negative or zero, then $\sum_{j=1}^{n}(n + 1 - j)\delta_j$ must be negative. This follows from the fact that for every δ_l that is positive by a certain amount, there is one or more δ_j that are negative by that amount with $j < l$. From the fact that the multiplier in the sum on the *RHS* is larger for $j < l$, it follows that the sum on the *RHS* is negative. With the *LHS* of the inequality being 0 and the *RHS* negative, a contradiction is again established. □

The correctness of Theorem 2.4 follows from Lemma 2.5 and Lemma 2.8.

Because of the complexity of the problem, it appears to be of interest to study heuristics that are based on simple priority rules. In the next section, some simple heuristics are proposed to solve this problem.

2.4 Simple Heuristics and Their Worst-Case Analyses

For the one-machine case $PD1 \mid\mid \sum w_j C_j$, it has been shown that Smith's *WSPT* (Weighted Shortest Processing Time first) rule (see [62]) is optimal. According to this rule, the jobs are scheduled in nondecreasing order of p_j/w_j, where p_j denotes the processing requirement of job j. It follows that when $w_j = 1$, it needs only to sequence the jobs in nondecreasing order of p_j. Usually, the unweighted version of the *WSPT* rule is named as *SPT* (Shortest Processing Time first). More or less by borrowing the greedy idea of the *SPT* rule, various heuristics that appear attractive in dealing with the total completion time objective have already been proposed in the literature. Most of these heuristics are greedy heuristics and generate a sequence of orders progressively one at a time.

Definition 2.2 *The Shortest Total Processing Time first (STPT) heuristic generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest total amount of processing over all m machines.*

Definition 2.3 *The Shortest Maximum Processing Time first (SMPT) heuristic generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest maximum amount of processing on any one of the m machines.*

Definition 2.4 *The Smallest Maximum Completion Time first (SMCT) heuristic first sequences the orders in nondecreasing order of p_{ij} on each machine $i = 1, 2, \dots, m$, then computes the completion time for order j as $C'_j = \max_{i=1}^{m}\{C_{ij}\}$, and finally schedules the orders in nondecreasing order of C'_j.*

Definition 2.5 *The Shortest Processing Time first applied to the machine with the largest load (SPTL) is a heuristic that generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest processing time on the machine that currently has the largest load.*

Definition 2.6 *The Earliest Completion Time first (ECT) heuristic generates a sequence of orders one at a time; each time it selects as the next order the one that would be completed the earliest.*

The *SMPT* and the *STPT* heuristics have been analyzed by Sung and Yoon [63]. Besides the *SMPT* and the *STPT* heuristics, Wang and Cheng [66] also studied the

SMCT heuristic. It seems that the *SPTL* heuristic and the *ECT* heuristic proposed above are new.

That such greedy heuristics or priority rules may perform quite poorly is well-known. The following examples illustrate instances in which three of the heuristics described above do not perform well.

Example 2.1 *Consider two machines and n orders that require 1 time unit on machine 1 and 0 on machine 2, n orders that require 1 time unit on machine 2 and 0 on machine 1, and $a \times n$ orders that require 1-ϵ on machine 1 and 1-ϵ on machine 2. The ratio*

$$\frac{\sum C_j(ECT)}{\sum C_j(OPT)}$$

is increasing in n and when $n \to \infty$ it becomes

$$\frac{a^2 + 4a + 2}{a^2 + 2a + 2}$$

This ratio reaches its maximum of $\sqrt{2}$ when $a = \sqrt{2}$. The optimal schedule in this case is to schedule the orders with the smallest total processing time first, i.e., the schedule is generated according to the STPT rule.

It is clear how to generalize this counterexample to an arbitrary number of machines m. The *ECT* rule will then perform even worse in comparison with the optimal rule. A counterexample to the *ECT* rule can also be constructed when all orders require the same total amount of processing. It should be noted that Example 2.1 also applies to the *SMCT* heuristic.

A counterexample to the *STPT* rule can be constructed as follows.

Example 2.2 *Consider 2 machines and n orders that require 1 unit on machine 2 and ϵ on machine 1, and $a \times n$ orders that require 1 unit on machine 1 only. By taking $a = (1 + \sqrt{5})/2$, one can obtain a value of the ratio of 1.618.*

This counterexample can be generalized to an arbitrary number of machines m. The *STPT* rule will then perform even worse in comparison with the optimal rule. Sung and Yoon [63] proved that for two machines

$$\frac{\sum C_j(STPT)}{\sum C_j(OPT)} \leq 2.$$

Wang and Cheng [66] generalized the above result to m machines and the bound is m. So, the worst-case ratio of the *STPT* rule for two machines is bounded above by 2 and Example 2.2 shows that the ratio can be as high as 1.618.

The performance of *SPTL* can be very bad. Consider the following example:

Example 2.3 *Consider 2 machines and 1 order that requires 0 time on machine 1 and A units of time on machine 2; in addition, there are B orders that require 1 time unit on both machines. If machine 1 is considered at $t = 0$, and let $A \gg B$, then the ratio becomes $(B + 1)$, which can be arbitrarily large. However, if at $t = 0$ machine 2 is considered, then the result is optimal.*

Theorem 2.9 *For $PD \mid\mid \sum C_j$,*

$$\frac{\sum C_j(ECT)}{\sum C_j(OPT)} \leq m.$$

Proof: Let $p_j = \max_{1 \leq i \leq m}\{p_{ij}\}$; let S_{ECT} denote the schedule generated by *ECT* and let S_{OPT} denote an optimal schedule. Furthermore, without loss of generality, it can be assumed that the orders are labeled in such a way that

$$p_1 \leq p_2 \leq \cdots \leq p_n.$$

First of all, it needs to be shown that

$$C_j(S_{ECT}) \leq \sum_{k=1}^{j} p_k, j = 1, 2, \ldots, n. \tag{2.16}$$

Clearly, the first order scheduled in S_{ECT} must be the one with the shortest maximum processing time. Thus,

$$C_1(S_{ECT}) = p_1.$$

Suppose that there exists a smallest position $j^*(1 < j^* < n)$ of the orders scheduled in S_{ECT}, such that

$$C_{j^*-1}(S_{ECT}) \leq \sum_{k=1}^{j^*-1} p_k,$$

but

$$C_{j^*}(S_{ECT}) > \sum_{k=1}^{j^*} p_k.$$

Then,

$$C_{j^*}(S_{ECT}) - C_{j^*-1}(S_{ECT}) > p_{j^*}.$$

So the maximum processing time of the $(j^*)^{th}$ order in S_{ECT} is larger than p_{j^*}. It follows that there exists at least one order with a subscript $l(2 \leq l \leq j^*$ and $p_l \leq p_{j^*})$ that was scheduled after position j^* in S_{ECT}. However, if order l is scheduled in position j^* in S_{ECT}, then $C_{j^*}(S_{ECT})$ will be smaller. This leads to a contradiction, since the *ECT* heuristic always chooses as the next order the one that would be completed the earliest.

From (2.16), it follows that

$$\sum_{j=1}^{n} C_j(S_{ECT}) \leq \sum_{j=1}^{n} \sum_{k=1}^{j} p_k. \tag{2.17}$$

Now, let $[j]$ be the order position in S_{OPT}. Then,

$$\begin{aligned} C_{[j]}(S_{OPT}) &= \max_{1\leq i\leq m}\left\{\sum_{k=1}^{j} p_{i[k]}\right\} \geq \sum_{k=1}^{j}\left(\sum_{i=1}^{m} p_{i[k]}\right)/m \\ &\geq \sum_{k=1}^{j}\max_{1\leq i\leq m}\{p_{i[k]}\}/m. \end{aligned} \tag{2.18}$$

Therefore,

$$\sum_{j=1}^{n} C_j(S_{OPT}) \geq \sum_{j=1}^{n}\sum_{k=1}^{j}\max_{1\leq i\leq m}\{p_{i[k]}\}/m \geq \sum_{j=1}^{n}\sum_{k=1}^{j} p_k/m. \tag{2.19}$$

The last "$\geq$" is due to the *SPT* rule.

Hence, the result follows from (2.17) and (2.19). □

2.5 Empirical Analysis

This section describes an empirical analysis of the heuristics for the $PDm \mid\mid \sum C_j$ problem. Due to the NP-hardness of the problem for $m \geq 3$ ($m = 2$ remains open), it is not likely to produce optimal solutions for the problem within limited running time by any exact algorithms. Thus, to evaluate the performance of these heuristics, a Tabu Search routine (Glover [21, 22]) is applied to improve on the results obtained by these heuristics; the Tabu Search routine basically serves as a tool for measuring the effectiveness of the five heuristics that are being analyzed.

2.5.1 Schedule Generation by Heuristics

Each one of the five heuristics is applied to every instance being considered and the five schedules generated are compared with one another. Implementing the five heuristics is relatively easy. Through a sorting algorithm, both *STPT* and *SMPT* can be implemented to run in $O(mn + n\lg n)$ time, and *SMCT* can be implemented to run in $O(mn\lg n)$ time. Both *SPTL* and *ECT* can be implemented in a natural way to run in $O(mn^2)$ time. Each heuristic produces a schedule and the best one of the five schedules generated is fed into the Tabu Search for postprocessing.

2.5.2 Postprocessing with Tabu Search

Tabu Structure In many applications of Tabu Search, reverse moves resulting in recently visited solutions are prohibited. However, if a short-term tabu list is used, it may be hard to avoid cycling. Surveys on the application of Tabu Search in intelligent scheduling systems can be found in Zweben & Fox [69] and Barnes et al. [3].

In what follows, a tabu structure that uses long-term memory (Glover and Laguna [23]) is proposed. The idea is to encode the solutions in an informative but concise way so that the representations describe the solution accurately and can be stored in a tabu list without taking too much memory. In addition, the representations are kept throughout until the Tabu Search procedure stops.

To represent a solution in a concise way, a schedule is encoded by using a data structure that consists of four data fields: 1) Cost of the schedule; 2) index of the first order in the sequence; 3) index of the middle order in the sequence; 4) index of the last order in the sequence.

It is possible that two different schedules have the same four data entries, but the probability of this occurring is small. If the four data entries of two different schedules are the same, then the two schedules are considered the same. Using such a representation, cycling definitely will not occur, but there is a small probability that some solutions will not be visited at all.

It should be noted that the above settings are somewhat similar to the tabu cycle method (Glover and Laguna [23]) which is based on short-term memory. For an implementation of the tabu cycle method, the reader is referred to Laguna [37].

Neighborhood Generating Mechanism A neighborhood generating mechanism defines a set of solutions that are to be explored by a local search procedure imbedded in Tabu Search.

Given a schedule S as shown in Figure 2.2(a), a so-called *Adjacent Subsequence Interchange (ASI)* as described in Figure 2.2 (b) generates a move from S to a neighboring solution of S. A so-called *Subsequence Reversal and Interchange (SRI)* also generates a move to a neighboring solution, see Figure 2.2 (c).

Each move is specified by three parameters: k_1, k_2 and k_3, where the positions k_1 and k_2 are the start and end positions of the segment that has to be transposed, while position k_3, which lies outside the segment $[k_1, k_2]$, is the new position in front of which the segment has to be moved to. Thus, these three position parameters define a neighborhood with about $O(n^3)$ distinct solutions for both *ASI* and SRI. To make the neighborhood size smaller so that the tabu search will run faster, it may be imposed that $k_2 - k_1 \leq 6$. Thus, the neighborhood size becomes $O(n^2)$. For any schedule S, the neighborhoods generated by *ASI* and *SRI* are denoted by $N_{ASI}(S)$ and $N_{SRI}(S)$, respectively.

The dominance result described in Lemma 2.3 is incorporated in the generation of $N_{ASI}(S)$ and $N_{SRI}(S)$ in order to prune the moves that result in solutions that can never be optimal. Thus, the number of solutions in $N_{ASI}(S)$ and $N_{SRI}(S)$ to be considered can be reduced.

Tabu Search Procedure The Tabu Search procedure operates as follows.

The Tabu Search Procedure

Step 1: $\{$*Initialization*$\}$

Select the best schedule S generated by *STPT*, *SMPT*, *SMCT*, *SPTL*, and *ECT*.

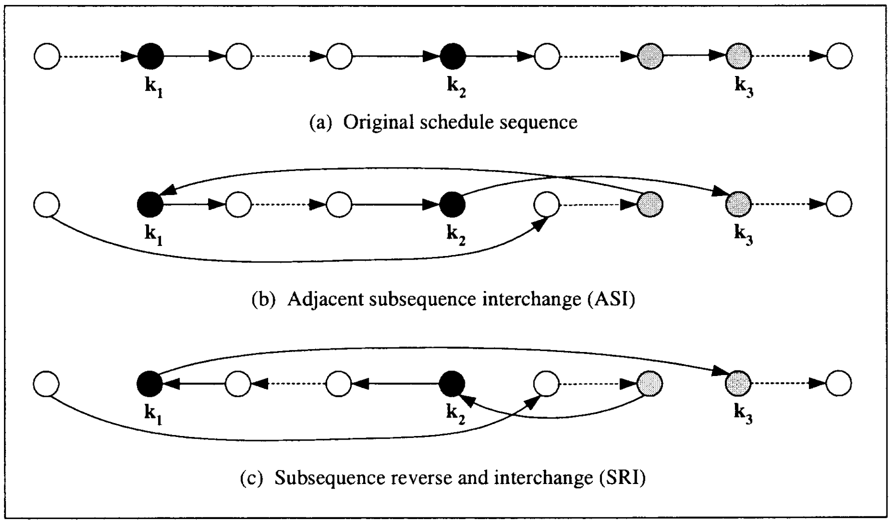

k1
k2
k3
(a) Original schedule sequence
(b) Adjacent subsequence interchange (ASI)
(c) Subsequence reverse and interchange (SRI)

Figure 2.2 Neighborhood generating operators.

Let $S_b \leftarrow S$; Set tabu list Γ to be empty.

Step 2: $\{$*Tabu Search*$\}$

Explore all the solutions in $N_{ASI}(S)$ and $N_{SRI}(S)$, and choose the best solution S_b' that is not in tabu list Γ.

If S_b' is better than S_b, then let $S_b \leftarrow S_b'$.

Add S into tabu list Γ; Let $S \leftarrow S_b'$.

Step 3: $\{$*Output*$\}$

If stop criterion is not met yet, goto Step 2. Otherwise, output S_b.

In the experiment, the stopping criterion is 1000 non-improving iterations in Step 2. Aspiration criterion and intensification/diversification mechanisms are not incorporated in the above tabu search procedure, since the purpose of using tabu search here is just to evaluate the quality of the results produced by the heuristics. Using these more intelligent mechanisms, the tabu search would perform better, but this is not the focus of this chapter. Readers who are interested in these techniques are referred to Glover and Laguna [23].

The running time of the Tabu Search algorithm is dominated by Step 2, each iteration of which examines $O(n^2)$ solutions in the neighborhoods $N_{ASI}(S)$ and $N_{SRI}(S)$. Since each exploration takes $O(mn)$ time to compute the objective cost of a solution and $O(1)$ time to check if a solution is in the tabu list Γ, each iteration of Step 2 takes $O(mn^3)$ time.

2.5.3 Generation of Problem Instances

For each problem size with $n = 20, 50, 100, 200$ orders and $m = 2, 5, 10, 20$ machines, 30 instances are randomly generated using a factor called *order diversity.* The order diversity κ is used to characterize the number of product types each order requests. The following three cases of order diversity are considered:

$\kappa = 2$: In problem instances 1 to 6 each order requests 2 different product types.

$\kappa = m$: In problem instances 7 to 12 each order requests the maximum number of different product types, namely m; m is the number of machines.

$\kappa = r$: In problem instances 13 to 30 each order requests a random number (r) of different product types; r is randomly generated from the uniform distribution $[1, m]$.

When the number of product types, l_j, for each order j is determined, l_j machines are chosen randomly. For each machine i that is selected, an integer processing time p_{ij} is generated from the uniform distribution $[1, 100]$. In total, $4 \times 4 \times 30 = 480$ instances are generated.

2.5.4 Experimental Results and Analyses

The algorithms are implemented in C++ with STL (Standard Template Library). The running environment is based on the RedHat Linux 7.0 operating system; the PC used was a Pentium II 400Mhz with 128MB RAM.

Table 2.1 The Percentage that Each Heuristic Performs the Best

Heuristic	*Percentage*
STPT	6.0
SMPT	0
SMCT	0
SPTL	6.0
ECT	88.0

For tables of detailed results produced by the algorithms for the generated problem instances, the reader is referred to Leung, Li, and Pinedo [48] or the URL http://web.njit.edu/~leung/pd-sumcj/tables.pdf. In each one of the tables, the results concerning the five heuristics appear in columns 3 to 7. The remaining columns are:

m:	The number of machines.
$H_{\min}$:	The name of the heuristic that yields the best result.
TS:	The result obtained with Tabu Search.
Imp:	The improvement obtained with Tabu Search (as a percentage).
$T_h(Sec)$:	The total running time (in seconds) of the five heuristics.
$T_{ts}(Sec)$:	The running time (in seconds) of Tabu Search.

Table 2.1 aggregates the results from the tables of detailed results. It shows that *ECT* is the best heuristic. Both *STPT* and *SPTL* produce the best schedule for some instances. However, neither *SMPT* nor *SMCT* produces the best schedule in any instance.

The detailed results also show that all instances for which *STPT* performs the best occur when $m = 2$, while the instances for which *SPTL* performs the best are distributed over all values of m. When there are more than 2 machines, *ECT* tends to be the best, and *SPTL* tends to be the second best. The differences between *ECT* and the other heuristics, especially the first three, can be quite substantial (often more than 5 percent; especially when the number of machines is large).

The average costs in Table 2.2 to Table 2.4, respectively, show that for $\kappa = 2$, the objective function decreases when m increases; for $\kappa = m$, the objective function increases when m increases; for $\kappa = r$, the change in the objective costs is not so clear.

With regard to the average performance, Table 2.2 to Table 2.4 also show that *ECT* performs better than all other heuristics. The improvement obtained by the Tabu Search is not that much. The tables also reveal that *SMPT* is the worst heuristic, while *SMCT* is slightly better than *SMPT*. *STPT* and *SPTL* are better than *SMPT* and *SMCT*. Therefore, the average performance is also consistent with Table 2.1.

Table 2.2 to Table 2.4 also show that, when $\kappa = 2$, the percentage improvement obtained through Tabu Search over the best result from the five heuristics decreases if m increases; when $\kappa = m$ or $\kappa = r$, this percentage tends to increase when m increases (even though for large n and m Tabu Search fails to provide much of an improvement because of the large solution spaces). This is consistent with the fact that the performance ratios of the heuristics become worse when m increases.

2.6 Concluding Remarks

This chapter focused on the fully dedicated order scheduling model with the objective of minimizing the total (unweighted) completion time. One of the main results in this chapter was the proof that $PDm \mid\mid \sum C_j$ is strongly NP-hard for every $m \geq 3$. Due to this, two new heuristics (*ECT* and *SPTL*) were proposed for solving the problem, and their worst-case performance was analyzed. In addition, the chapter presented an empirical analysis of the two heuristics together with other three heuristics (*STPT*, *SMPT*, and *SMCT*) appeared in the literature. The experimental results showed that

Table 2.2 Comparison of Average Costs when $\kappa = 2$

n	m	*STPT*	*SMPT*	*SMCT*	*SPTL*	*ECT*	*TS*	*Imp*
	5	5140	5252	4795	4754	4539	4389	3.35
20	10	3904	4082	3408	3622	3161	3118	1.38
	20	3246	3156	2543	2809	2411	2401	0.43
	5	26962	28081	25002	24315	23193	22831	1.58
50	10	16684	17079	14291	15396	12913	12808	0.83
	20	11455	11621	9356	10953	8447	8431	0.20
	5	97028	99304	92602	89609	85675	84288	1.63
100	10	60583	62130	50860	54137	46273	46209	0.13
	20	38073	38177	30852	35315	27197	27190	0.02
	5	384440	398887	364984	350516	341139	334658	1.93
200	10	212608	215531	187304	190922	170692	170547	0.08
	20	127751	127392	107108	119232	93620	93607	0.02

Table 2.3 Comparison of Average Costs when $\kappa = m$

n	m	*STPT*	*SMPT*	*SMCT*	*SPTL*	*ECT*	*TS*	*Imp*
	2	9719	9814	9662	9525	9283	9193	0.97
	5	12006	12057	11909	11662	11338	11166	1.52
20	10	12934	13114	12515	12418	12031	11808	1.88
	20	13142	13410	13187	12758	12314	12031	2.37
	2	50672	51973	51382	51386	49689	48817	1.55
	5	63865	66194	64880	61930	60249	59114	1.92
50	10	68298	72787	70292	67787	65094	63865	1.93
	20	71325	73682	72242	70094	67394	66227	1.78
	2	209003	217536	214953	209384	206885	202199	1.83
	5	245043	253867	250694	241537	232949	228461	1.95
100	10	263542	274975	269076	258865	248448	245098	1.38
	20	272267	280485	275442	265886	256651	253466	1.27
	2	827466	848915	839999	829966	820013	799041	2.22
	5	961302	1001502	987641	951628	917933	900094	1.97
200	10	1004000	1062621	1043997	998574	965789	952002	1.45
	20	1038045	1092310	1071347	1026776	993572	985501	0.82

Table 2.4 Comparison of Average Costs when $\kappa = r$

n	m	*STPT*	*SMPT*	*SMCT*	*SPTL*	*ECT*	*TS*	*Imp*
	2	6341	6500	6203	6262	5996	5887	1.58
	5	6904	6822	6416	6295	6019	5917	1.79
20	10	6661	6914	6522	6064	5811	5684	2.12
	20	7456	8218	7562	6979	6704	6544	2.43
	2	35224	36975	35722	35022	34217	33206	2.68
	5	36075	38168	36148	33181	32493	31433	3.38
50	10	34765	38639	36155	32066	30864	30056	2.65
	20	35194	39876	37354	33121	31951	30910	3.29
	2	150013	155842	151452	148438	145675	141165	3.02
	5	128211	138648	132673	120410	118702	114338	3.71
100	10	124160	136125	130186	114024	111213	107923	3.03
	20	124063	141346	134570	115521	110666	108286	2.17
	2	558391	583815	569089	561344	551590	531680	3.36
	5	469938	520905	498128	443399	445451	425430	3.99
200	10	451059	511691	490323	422835	416597	405359	2.75
	20	453159	520763	497481	425083	411549	404229	1.81

ECT is the best one among all the five heuristics. Thus, in practice, *ECT* would be a good choice for solving real-life problem instances, with regard to its speed and solution quality.

In the literature, the weighted version of *STPT*, *SMPT*, and *SMCT* have been shown to have the same worst-case performance ratio as their unweighted version. It would be interesting to investigate if *ECT* and *SPTL* can be generalized to solve the weighted problem $PD \mid\mid \sum w_j C_j$. In Chapter 4, it will be shown that the two heuristics do have their generalized versions for the weighted problem. In addition, with additional constraints on processing times, some interesting observations on the weighted versions of the heuristics will also be explored in Chapter 4.

For the fully dedicated case, some other unweighted objectives (e.g., the due date related objectives) will be studied in Chapter 3.

CHAPTER 3

THE FULLY DEDICATED CASE – DUE DATE RELATED OBJECTIVES

3.1 Introduction

This chapter still examines the fully dedicated case of order scheduling. However, the objectives of interest are related to due dates.

With regard to the due date related objectives, most of the past work on the fully dedicated case has focused on the minimization of the number of late orders. Wagneur and Sriskandarajah [65] showed that $PD2 \mid\mid \sum U_j$ is NP-hard in the ordinary sense. Cheng and Wang [10] showed that there exists a pseudo-polynomial time algorithm for every fixed m. Ng, Cheng, and Yuan [55] showed that when the number of machines is arbitrary the problem becomes strongly NP-hard; they showed that the more restricted problem $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$ is strongly NP-hard as well.

In this chapter, the attention is focused on various due date related objectives including the maximum lateness $L_{\max}$ and the number of late orders $\sum U_j$.

This chapter contains the results published in Leung, Li, and Pinedo [46]. The chapter is organized as follows. Section 3.2 considers some Min-Max objectives that are related to due dates. Section 3.3 analyzes a greedy algorithm for minimizing the total number of tardy orders with a common due date. It also describes a heuristic and an exact algorithm for minimizing the total number of tardy orders with distinct due dates. Section 3.4 presents an empirical analysis of the heuristic and the exact algorithm. Finally, Section 3.5 contains some concluding remarks.

3.2 Min-Max Objectives

The easiest objective to analyze is the maximum lateness objective. This objective can be minimized by the *EDD* (Earliest Due Date first) rule (see Jackson [32]); i.e., the next finished product is assigned to the customer with the earliest due date. Using an adjacent interchange argument, one can easily prove the following theorem.

Theorem 3.1 *The Earliest Due Date rule solves the problem $PD \mid\mid L_{\max}$.*

Suppose that, in a nonpreemptive environment, each order has a deadline $\bar{d}_j$ that *must* be adhered to (instead of a due date d_j) and the goal is to determine if there is a feasible schedule that meets all the deadlines. This problem can be solved by the *Earliest Deadline rule.*

The optimality of the *EDD* rule and the Earliest Deadline rule are special cases of a result for the more general problem $PD \mid prec \mid f_{\max}$. In this more general problem the processing of the orders are subject to precedence constraints and the objective function is the more general $f_{\max}$. Lawler [38] developed an algorithm based on (backwards) dynamic programming that solves the single machine version of this problem in polynomial time. Lawler's algorithm can be extended in such a way that it generates optimal solutions for the more general PD machine environment.

Theorem 3.2 *The problem $PD \mid prec \mid f_{\max}$ can be solved in $O(n^2)$ time.*

This problem with the $f_{\max}$ objective includes some interesting special cases. For example, consider the problem in which order j has both a due date d_j and a deadline $\bar{d}_j$ ($\bar{d}_j \geq d_j$). Suppose the objective is to minimize $L_{\max}$ under the condition that each order must be completed before its deadline. This is equivalent to order j being subject to the penalty function $f(C_j)$, where

$$\begin{aligned} f_j(C_j) &= C_j - d_j \quad && \text{if } C_j \leq \bar{d}_j \\ f_j(C_j) &= \infty && \text{if } C_j > \bar{d}_j \end{aligned}$$

Applying Lawler's algorithm with the cost functions described above solves the problem $1 \mid \bar{d}_j \mid L_{max}$ and this algorithm can be adapted in such a way that it can be applied to $PD \mid \bar{d}_j \mid L_{\max}$.

When the orders are not all available at time $t = 0$, the problem becomes significantly more complex. The Earliest Due Date rule is not optimal when the orders have different release dates. The NP-hardness of $1 \mid r_j \mid L_{\max}$ implies the NP-hardness of $PD1 \mid r_j \mid L_{\max}$.

However, if preemptions are allowed on all product types at any time, then the $PD \mid r_j, prmp \mid L_{\max}$ problem is easy.

Theorem 3.3 *The preemptive Earliest Due Date rule solves the problem* $PD \mid r_j, prmp \mid L_{\max}$.

If the orders have different release dates, then the preemptive *Earliest Deadline* rule also determines if there is a schedule that meets all the deadlines.

3.3 Minimizing the Total Number of Late Orders $\sum U_j$

Consider now the same problem, but with the objective function $\sum U_j$, i.e., the problem $PD \mid\mid \sum U_j$. As mentioned before, this problem is strongly NP-hard. However, there are some special cases that are solvable in polynomial time.

First, $PDm \mid p_{ij} \in \{0, 1\} \mid \sum U_j$ can be solved by an $O(nP^m)$ algorithm (Cheng and Wang [10]). Since $P = \max_{1 \leq i \leq m}\{\sum_{j=1}^{n} p_{ij}\}$, the algorithm only takes $O(n^{m+1})$ time, which is polynomial for fixed m. Actually, if $p_{ij} \leq c$ for all i and j, where c is an integer, the problem is still polynomially solvable by the same algorithm. However, for large values of c and m, the algorithm will take a very long time.

Second, consider the special case in which the processing times of the orders satisfy the following "agreeability" conditions, which is equivalent to a form of *order* *dominance*. The orders can be ranked in such a way that

$$p_{i1} \leq p_{i2} \leq \cdots \leq p_{in}, \quad i = 1, 2, \ldots, m.$$

So order 1 requires the least amount of time on each one of the m machines and order n requires the most time on each one of the m machines. In this case an optimal schedule can be generated by scheduling the orders according to *EDD*. If an order is scheduled and its completion on any one of the m machines is after its due date (i.e., if the maximum of its completion time over all m machines is larger than its due date), then the order in the partial schedule that has the longest processing time on each one of the machine is deleted. In what follows this algorithm is referred to as the Revised Hodgson-Moore (*RHM*) algorithm. The following theorem is presented without proof.

Theorem 3.4 *The RHM Algorithm solves $PD \mid\mid \sum U_j$ when p_{ij} are agreeable, i.e., $p_{i1} \leq p_{i2} \leq \ldots \leq p_{in}$ for $i = 1, \ldots, m$.*

Actually, the agreeability conditions on the processing times described above are relatively strong. Somewhat weaker conditions would already assure that the *RHM* yields an optimal schedule for $PD \mid\mid \sum U_j$:

Theorem 3.5 *If in the application of RHM whenever an order has to be deleted, there is an order that has the longest processing time on each one of the machines, then RHM yields an optimal schedule.*

Proof: The proof is an adaptation of the one given for the Hodgson-Moore algorithm by Pinedo [57]. Assume without loss of generality that $d_1 \leq d_2 \leq \ldots \leq d_n$. Let $\mathcal{O}_k$ be a subset of orders $\{1, 2, \ldots, k\}$, and it satisfies the following two conditions:

(1) Among all subsets of $\{1, 2, \ldots, k\}$, $\mathcal{O}_k$ has the maximum number of orders, say N_k, completed by their due dates.

(2) Among all subsets of $\{1, 2, \ldots, k\}$ that have N_k orders delivered on time, the orders in the subset $\mathcal{O}_k$ require the smallest total processing time on each machine $i = 1, 2, \ldots, m$.

With the above two conditions, it is clear that $\mathcal{O}_n$ is the set of orders completed by their due dates in an optimal schedule. The proof that the *RHM* algorithm leads to $\mathcal{O}_n$ is by induction. For the base case $k = 1$, it is true that the *RHM* algorithm yields $\mathcal{O}_k$ satisfying the two conditions. The inductive hypothesis is that the algorithm constructs a set $\mathcal{O}_k$ satisfying the two conditions. Now it needs to be shown that, starting out with $\mathcal{O}_k$, the algorithm yields in the $(k+1)^{th}$ iteration a set $\mathcal{O}_{k+1}$ that satisfies the two conditions for $k + 1$. There are two cases to be considered:

Case 1: Order $k + 1$ is added to $\mathcal{O}_k$ and it is completed by its due date. Now $N_{k+1} = N_k + 1$ and $\mathcal{O}_{k+1} = \mathcal{O}_k \cup \{k + 1\}$. Clearly, the two conditions are satisfied.

Case 2: Order $k+1$ is added to $\mathcal{O}_k$ and it is tardy. By the inductive hypothesis, N_k is the maximum number of orders to be completed by their due dates among orders $\{1, 2, \ldots, k\}$. In addition, the orders in $\mathcal{O}_k$ have the smallest total processing time on each machine $i = 1, 2, \ldots, m$. It follows that $N_{k+1} = N_k$. Thus, adding order $k+1$ to set $\mathcal{O}_k$ does not increase the number of orders completed by their due dates. Let j^* be the order in $\mathcal{O}_k \cup \{k + 1\}$, such that it has the longest processing time on each one of the m machines. Clearly, among all N_k-order subsets of $\{1, 2, \ldots, k, k+1\}$, the orders in $\mathcal{O}_k \cup \{k+1\} \setminus \{j^*\}$ have the smallest total processing time on each machine $i = 1, 2, \ldots, m$. The two conditions hold.

From the above inductive proof, the result follows. □

It is clear that $PD \mid r_j \mid \sum U_j$ is strongly NP-hard due to the strong NP-hardness of $1 \mid r_j \mid \sum U_j$ (See Garey and Johnson [19]). However, if it satisfies $r_j < r_k \Rightarrow d_j \leq d_k$, the following result can be shown:

Theorem 3.6 *For $PD \mid r_j < r_k \Rightarrow d_j \leq d_k \mid \sum U_j$, if in the application of* RHM, *whenever an order has to be deleted, there is one order that has the longest processing time on each one of the machines, then the* RHM *algorithm yields an optimal schedule.*

Proof: Kise, Ibaraki, and Mine [35] presented a proof for the validity of applying the Hodgson-Moore algorithm to solve $1 \mid r_j < r_k \Rightarrow d_j \leq d_k \mid \sum U_j$. Their proof can be adapted in a way similar to the proof of the previous theorem to show the correctness of the result. □

The results presented with regard to machine dominance and order dominance are useful in the implementation of heuristics. For example, if a one-pass heuristic is used that is inserting orders in a forward manner, then order dominance as well as machine dominance should be checked at every step. Whenever an order has to be deleted from the partial schedule, it has to be checked whether or not any one order dominates all others in the partial schedule. If one order dominates, then that order has to be deleted. Each time an order has been deleted from the partial schedule, machine dominance has to be checked with regard to that set of orders that includes the remaining orders in the partial schedule plus the set of orders that still have to be considered. When a machine is being dominated by another machine with regard to this (smaller) set of orders, then that machine may be disregarded in the remaining steps of the heuristic.

The results presented above may prove useful in the implementation of exact algorithms as well. For example, suppose that when the *RHM* procedure indicates that one order in the partial schedule should be deleted, and there are two orders that dominate all others in the partial schedule, i.e., they have processing times that are longer than any other order on each one of the m machines. However, one of the two orders has on some of the machines the longest processing time and the other order has on the remaining machines the longest processing time. An exact procedure should now branch out and consider two partial schedules. In one of the partial schedules one of the two orders is deleted, and in the other partial schedule the other order is deleted.

3.3.1 A Greedy Algorithm for the Common Due Date Case

Consider now the special case of $PD \mid\mid \sum U_j$ in which all orders have the same due date, that is, $d_j = d$. This problem is denoted by $PD \mid d_j = d \mid \sum U_j$. The greedy heuristic presented in this section is based on the observation that there exists a close connection between $PD \mid d_j = d \mid \sum U_j$ and the **Multiset Multicover Problem** (*MSMC*) which can be described as follows (see [60, 64]):

Definition 3.1 (The Multiset Multicover Problem) *Let $\mathcal{U} = \{e_1, e_2, \ldots, e_m\}$ be a base set; $\mathcal{C} = \{S_j : j = 1, 2, \ldots, n\}$, where $e_l \in S_j \Rightarrow e_l \in \mathcal{U}$. However, S_j can be a multiset so that any element $e_l \in S_j$ can appear more than once in S_j. Let $\mathcal{N}(e_l, S_j)$ denote the number of times that element e_l appears in S_j. Each element $e_l \in \mathcal{U}$ needs to be covered b_l times. Each multiset S_j can be selected at most once. The objective is to pick a minimum number of multisets from $\mathcal{C}$, such that each element e_l is covered at least b_l times, for $l = 1, 2, \ldots, m$.*

Clearly, the *MSMC* problem is strongly NP-hard, since the strongly NP-hard Set Cover Problem (see Garey and Johnson [19]) is a special case.

In what follows it will be shown that, $PD \mid d_j = d \mid \sum U_j$ can be converted into an *MSMC* problem that is equivalent; this implies that any known algorithm for the *MSMC* problem can be used for $PD \mid d_j = d \mid \sum U_j$ as well.

Given any instance $\mathbf{I}_{PD}$ of $PD \mid d_j = d \mid \sum U_j$ with m machines, n orders, integer processing times, and integer due date d, one can construct an *MSMC* instance $\mathbf{I}_{MM}$ as follows:

1) Let the number of elements in the base set $\mathcal{U}$ be m.

2) For each order $j \in \mathcal{O} = \{1, 2, \ldots, n\}$, construct a multiset S_j as follows: For each $i = 1, 2, \ldots, m$, let $\mathcal{N}(e_i, S_j) = p_{ij}$ be the number of times element e_i appears in S_j. Let $\mathcal{C} = \{S_j : j = 1, 2, \ldots, n\}$.

3) Let the coverage requirement b_i be equal to $\sum_{j \in J} p_{ij} - d$ for element e_i, $i = 1, 2, \ldots, m$. Note that if $b_i \leq 0$, it means that all the orders can be completed

on machine i before d. Thus, machine i can be ignored. Due to this reason, it is assumed that $b_i > 0$ for any $i = 1, 2, \ldots, m$.

Clearly, the above transformation takes $O(nm)$ time, which is polynomial. With this transformation, the following result is claimed:

Lemma 3.7 *If $\mathcal{C}^* = \{S_{j_1}, S_{j_2}, \ldots, S_{j_{|\mathcal{C}^*|}}\} \subseteq \mathcal{C}$ is a cover produced by an algorithm $\mathbb{A}$ for the instance $\mathbf{I}_{MM}$ of the* MSMC *problem, then there exists a schedule π for the instance $\mathbf{I}_{PD}$ of $PD \mid d_j = d \mid \sum U_j$ in which $\mathcal{O}^* = \{j_1, j_2, \ldots, j_{|\mathcal{C}^*|}\}$ is the set of late orders and $\mathcal{O} \setminus \mathcal{O}^*$ is the set of early orders. In addition, $|\mathcal{C}^*| = \sum_{j=1}^{n} U_j(\pi)$.*

Proof: Since $\mathcal{C}^* = \{S_{j_1}, S_{j_2}, \ldots, S_{j_{|\mathcal{C}^*|}}\} \subseteq \mathcal{C}$ is a cover for the *MSMC* instance $\mathbf{I}_{MM}$, it follows that for each $i = 1, 2, \ldots, m$,

$$\begin{aligned} \sum_{j \in \mathcal{C}^*} \{\mathcal{N}(e_i, S_j) : e_i \in S_j\} \geq b_i \;&\Rightarrow\; \sum_{j \in \mathcal{O}^*} p_{ij} \geq b_i \\ &\Rightarrow\; \sum_{j \in \mathcal{O} \setminus \mathcal{O}^*} p_{ij} \leq \sum_{j \in \mathcal{O}} p_{ij} - b_i = d. \end{aligned}$$

Thus, a schedule π can be constructed from $\mathcal{C}^*$ for the $PD \mid d_j = d \mid \sum U_j$ instance $\mathbf{I}_{PD}$, in which the orders in $\mathcal{O} \setminus \mathcal{O}^*$ are completed by the common due date d, while the orders in $\mathcal{O}^*$ are late. Now, it is easy to see that $|\mathcal{C}^*| = \sum_{j=1}^{n} U_j(\pi)$. □

By Lemma 3.7, the following lemma can be shown:

Lemma 3.8 *Any approximation algorithm for the MSMC Problem can be applied to solve $PD \mid d_j = d \mid \sum U_j$ with integer processing times and integer d, with the approximation ratio being preserved.*

Proof: By Lemma 3.7, if $\mathbb{A}$ is an exact algorithm which solves $\mathbf{I}_{MM}$, then it can also be used to generate an optimal solution for $\mathbf{I}_{PD}$. In addition, the optimal objective costs of the two instances are equal. On the other hand, if $\mathbb{A}$ is an approximation algorithm for $\mathbf{I}_{MM}$, it will also result in the same objective cost for $\mathbf{I}_{PD}$. It follows that the approximation ratio is preserved. □

Based on the above observation, it is of interest to apply known approximation algorithms for the *MSMC* problem to the $PD \mid d_j = d \mid \sum U_j$ problem. Rajagopalan and Vazirani [60] presented a greedy algorithm for solving the weighted *MSMC* problem, in which each multiset S_j has a weight w_j and the objective is to minimize the total weight of the picked multisets. One can consider element e_i as being "alive" if it occurs fewer than b_i times in the selected multisets. Let $\mathcal{A}$ denote the set of elements that are still alive; let C' denote the set of multisets that have not been selected yet. The greedy algorithm works as follows: In each iteration, the algorithm picks, among C', a multiset S_{j^*} such that

$$S_{j^*} = \arg\min_{S_j \in C'} \left\{ w_j / \sum_{e_i \in \mathcal{A}} \min\{\mathcal{N}(e_i, S_j), b'_i\} \right\},$$

where b'_i is the residual coverage requirement of e_i; it is initialized as b_i and is decremented by $\mathcal{N}(e_i, S)$ each time a set S is selected. If any element e_i is covered by the selected multisets at least b_i times, it is not "alive" any more. The algorithm terminates when there are no more "alive" elements left. Rajagopalan and Vazirani [60] showed that the approximation ratio of this greedy algorithm is

$$\mathcal{H}\left(\max_{1 \le j \le n} \left\{ \sum_{e_i \in S_j} \mathcal{N}(e_i, S_j) \right\} \right),$$

where $\mathcal{H}(k) \equiv \sum_{i=1}^{k} \frac{1}{i}$.

Let $x_j \in \{0, 1\}$ be the decision variable to indicate that S_j is "selected" if $x_j = 1$, otherwise "unselected". It is noticed that the coverage constraint for element e_i is:

$$\sum_{j=1}^{n} x_j \cdot \mathcal{N}(e_i, S_j) \ge b_i.$$

Thus, let c_i be the greatest common divisor (*GCD*) among

$$\mathcal{N}(e_i, S_1), \mathcal{N}(e_i, S_2), \ldots, \mathcal{N}(e_i, S_n), b_i,$$

then for any settings of $x_j, j = 1, 2, \ldots, n$,

$$\sum_{j=1}^{n} x_j \cdot \frac{\mathcal{N}(e_i, S_j)}{c_i} \geq \frac{b_i}{c_i} \;\Rightarrow\; \sum_{j=1}^{n} x_j \cdot \mathcal{N}(e_i, S_j) \geq b_i.$$

Therefore, with a preprocessing procedure to "condense" the constraint data, the approximation ratio of the greedy algorithm can be rewritten as

$$\mathcal{H}\left(\max_{1 \leq j \leq n} \left\{ \sum_{e_i \in S_j} \frac{\mathcal{N}(e_i, S_j)}{c_i} \right\}\right).$$

Based on this, with a procedure of preprocessing the constraint data, Rajagopalan and Vazirani's greedy algorithm can be adapted to solve $PD \mid d_j = d \mid \sum U_j$ as an equivalent unweighted *MSMC*. The algorithm is presented as follows:

The Greedy Algorithm for $PD \mid d_j = d \mid \sum U_j$

Step 1: { *Initialization* }

Let $U_j = 0$ for each $j = 1, 2, \ldots, n$.

Let $c_i = GCD(p_{i1}, p_{i2}, \ldots, p_{in}, d)$ for each $i = 1, 2, \ldots, m$;

If $c_i > 1$, let $p_{ij} = p_{ij}/c_i$ for each $j = 1, 2, \ldots, n$.

Let $b_i = \sum_{j=1}^{n} p_{ij} - d/c_i$ for each $i = 1, 2, \ldots, m$.

Let $\mathcal{O} = \{1, 2, \ldots, n\}$ be the initial order set.

Let $\pi_e = \emptyset$ and $\pi_l = \emptyset$.

Let $\mathcal{M} = \{1, 2, \ldots, m\}$ be initial set of the indices of machines.

Step 2: {*Greedy Selection* }

If $|\mathcal{M}| = 0$, then goto Step 3.

Choose from $\mathcal{O}$ the order j^* such that $j^* = \arg\min_{j \in \mathcal{O}}\{1/\sum_{i \in \mathcal{M}} \min\{p_{ij}, b_i\}\}$; ties are broken arbitrarily.

$U_{j^*} = 1$; $\pi_l = \pi_l \cup \{j^*\}$; $\mathcal{O} = \mathcal{O} \setminus \{j^*\}$.

Update $b_i = b_i - p_{ij^*}$ for each machine $i \in \mathcal{M}$.

If any b_i becomes less than or equal to 0, then remove i from $\mathcal{M}$.

Repeat Step 2.

Step 3: $\{$*Algorithm Terminates*$\}$

Construct a schedule π:

Let $\pi_e = \mathcal{O}$ be the set of early orders, and let π_l be the set of late orders.

Output schedule π, and return $|\pi_l|$ as the number of late orders.

Note that the set $\mathcal{M}$ in the algorithm corresponds to the set $\mathcal{A}$ (the set of elements that are still alive).

To evaluate the running time of the above algorithm, it is necessary to consider how to compute $c_i = GCD(p_{i1}, p_{i2}, \cdots, p_{in}, d)$. Even though Euclid's *GCD* algorithm applies only to two arguments, c_i can be calculated by

$$GCD(p_{i1}, p_{i2}, \cdots, p_{in}, d) = GCD(p_{i1}, GCD(p_{i2}, \cdots, p_{in}, d))$$

in a recursive way. Let $a > b$, Euclid's algorithm for $GCD(a, b)$ runs in $O(\lg b)$ division steps (see [12]). If bit operations are not considered, then each division takes a constant time. In addition, it is reasonable to assume that all $p_{ij} \leq d$. Therefore, the m c_i's can be computed in $O(mn \lg d)$ time, which dominates the running time of Step 1. In each iteration of Step 2, it is clear that the running time is dominated by choosing from $\mathcal{O}$ the order j^* such that

$$j^* = \arg\min_{j \in \mathcal{O}} \left\{ 1 / \sum_{i \in \mathcal{M}} \min\{p_{ij}, b_i\} \right\},$$

which takes $O(nm)$ time. Step 2 runs at most n iterations. Thus, it takes $O(n^2 m)$ time. In Step 3, the output takes $O(n)$ time. It follows that the entire algorithm runs in $O(n^2 m + mn \lg d)$.

For any instance of $PD \mid d_j = d \mid \sum U_j$, let $\sum U_j(H)$ be the objective cost obtained by the above greedy heuristic, and let $\sum U_j(OPT)$ be the optimal objective cost. From the above discussion, the following result follows:

Theorem 3.9 *For $PD \mid d_j = d \mid \sum U_j$, if all p_{ij} and d are integers, then*

$$\frac{\sum U_j(H)}{\sum U_j(OPT)} \leq \mathcal{H}\left(\max_{1\leq j\leq n}\left\{\sum_{i=1}^{m}\frac{p_{ij}}{c_i}\right\}\right),$$

where $c_i = GCD(p_{i1}, p_{i2}, \ldots, p_{in}, d)$ for $i = 1, 2, \ldots, m$.

To see that the bound is attainable, it is easy to construct a scheduling example by generalizing the Set Cover example from Johnson [33].

As pointed out previously, Rajagopalan and Vazirani's greedy algorithm was designed for weighted *MSMC*. For the problem studied in this section, if each order has a weight w_j, it is easy to generalize the above greedy algorithm to solve $PD \mid d_j = d \mid \sum w_j U_j$, if all p_{ij} and d are integers. The only modification to the algorithm is that, the order to be chosen in each iteration is order j^* such that

$$j^* = \arg\min_{j\in\mathcal{O}}\left\{w_j / \sum_{i\in\mathcal{M}} \min\{p_{ij}, b_i\}\right\}.$$

It is clear that the approximation ratio of the generalized greedy algorithm for $PD \mid d_j = d \mid \sum w_j U_j$ remains unchanged. For $PD \mid d_j = d \mid \sum w_j U_j$, another observation is that it is in fact the dual problem of the Multidimensional 0-1 Knapsack Problem (*MKP*) with an arbitrary number of dimensions. Thus, the resolution methods for *MKP* also shed light on solving $PD \mid d_j = d \mid \sum w_j U_j$. For a very recent survey for the *MKP* problem, the reader is referred to Fréville [17].

For $PD \mid p_{ij} \in \{0, 1\}, d_j = d \mid \sum w_j U_j$, Ng, Cheng, and Yuan [55] presented a $(d + 1)$-approximation LP-rounding algorithm. However, if the generalized greedy algorithm is applied to solve this problem, its approximation ratio is at most $\mathcal{H}(m)$. Thus, if $m \leq e^d$, the approximation ratio of the greedy algorithm would be better than that of the LP-rounding heuristic. In fact, by the reduction shown previously, the $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum w_j U_j$ problem turns out to be a weighted Set Multicover (*SMC*) Problem, since the elements in each constructed set are unique but each element is required to be covered multiple times. Thus, any approximation algorithms for weighted *SMC* can be applied to solve $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum w_j U_j$ with the approximation ratio being preserved. Hochbaum [29] presented three LP-based ρ-approximation algorithms for weighted *SMC*, where $\rho = \max_{1 \le j \le n} \{\sum_{i=1}^{m} p_{ij}\}$. Clearly, these ρ-approximation algorithms can be applied to solve $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum w_j U_j$. Since $\mathcal{H}(\rho) < \rho$ for $\rho \ge 2$, it is easy to see that the approximation ratio of the generalized greedy algorithm is still better. Of course, the above comparisons remain effective for $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$.

3.3.2 A Heuristic for $PD \mid\mid \sum U_j$

Heuristics for the $PD \mid\mid \sum U_j$ problem with arbitrary processing times can be designed based on the ideas of the Hodgson-Moore [51] algorithm for $PD1 \mid\mid \sum U_j$ and the machine dominance and order dominance concepts introduced in the previous sections. Using the main idea behind the Hodgson-Moore algorithm the following heuristic can be designed for $PD \mid\mid \sum U_j$. The orders are put in schedule π_e according to *EDD*. Whenever an order j' that is put into the schedule is completed after its due date, one of the orders that are currently part of the schedule has to be taken out.

Selecting the order to be taken out is done based on a priority ranking system. In order to make sure that not more than one order has to be deleted from the schedule, it pays to keep a set of candidate orders $\mathcal{O}_c$ with the property that the removal of any one order in $\mathcal{O}_c$ from π_e ensures that the rest of the orders in π_e are completed before their due dates.

First of all, the tardy order j' itself is already a candidate order, since all the orders that precede j' in π_e can be completed before their due dates. For each order

$j \in \pi_e, j \neq j'$, if its removal from π_e enables order j' to be completed in time, then j becomes a candidate in $\mathcal{O}_c$; otherwise, j will not become a candidate. It is clear that $1 \leq |\mathcal{O}_c| \leq |\pi_e|$.

Secondly, for each candidate order $j \in \mathcal{O}_c$, a weighted sum of all its processing times p_{ij} on the m machines, denoted by $W(\mathcal{O}_c)_j$, has to be computed. The weight of machine i, denoted by ϖ_i, is a function of the current load on machine i, denoted by CL_i, and the future workload of machine i due to all the orders that still have to be considered, denoted by FL_i. A typical weight function can be

$$\varpi_i = \omega_1 \cdot CL_i + \omega_2 \cdot FL_i,$$

where ω_1 and ω_2 are the weights for CL_i and FL_i, respectively, for any $i = 1, \ldots, m$. With ϖ_i, the weighted sum of each candidate order $j \in \mathcal{O}_c$ is computed as

$$W(\mathcal{O}_c)_j = \sum_{i=1}^{m} \varpi_i \cdot p_{ij}.$$

Finally, the candidate order to be taken out is the one with the maximum weighted sum, i.e., order j^* such that

$$j^* = \arg\max_{j \in \mathcal{O}_c} \{W(\mathcal{O}_c)_j\}.$$

Without loss of generality it may be assumed that the orders are ranked in increasing order of their due dates, i.e., $d_1 \leq d_2 \leq \cdots \leq d_n$. The input of the heuristic is a set of orders, and the output of the heuristic is a sequence in which the orders are scheduled. Note that the sequence consists of two parts. The first part, namely π_e, consists of those orders that are completed on time; these orders are ordered according to *EDD*. The second part of the sequence, namely π_l, consists of the orders that are not completed on time; it does not matter in which order this subset appears in the sequence. The goal of the heuristic is to determine which orders have to be placed in π_e.

The above ideas are implemented in the following heuristic, which is referred to as the *Generalized Hodgson-Moore* heuristic or *GHM* heuristic.

The Generalized Hodgson-Moore Heuristic (*GHM*) for $PD \,||\, \sum U_j$

Step 1: {*Initialization*}

Sort the orders in ascending order of due dates; i.e., $d_1 \leq d_2 \leq \cdots \leq d_n$.

Let $\pi_e = \emptyset$, $\pi_l = \emptyset$.

$CL_i = 0$ and $FL_i = \sum_{j=1}^{n} p_{ij}$ for $i = 1, \ldots, m$.

Assign values to the weights ω_1 and ω_2.

Let $j = 1$.

Step 2: {*Generating a sequence*}

Include order j in π_e and compute C_{ij} for all i, i.e.,

$$C_{ij} = \sum_{k \in \pi_e} p_{ik}, \quad i = 1, \ldots, m.$$

For each machine $i = 1, 2, \ldots, m$, let $CL_i = CL_i + p_{ij}, FL_i = FL_i - p_{ij}$.

If $\max(C_{1j}, \ldots, C_{mj}) \leq d_j$, then go to Step 4;

Otherwise go to Step 3.

Step 3: {*Delete order j^* from π_e*}

Create set $\mathcal{O}_c$: for each order $k \in \pi_e$, if its removal from π_e results in all remaining orders in π_e being completed on time, then k is included in $\mathcal{O}_c$.

Compute $\varpi_i = \omega_1 \cdot CL_i + \omega_2 \cdot FL_i$ for $1 \leq i \leq m$.

Compute $W(\mathcal{O}_c)_k = \sum_{i=1}^{m} \varpi_i \cdot p_{ik}$ for all k in $\mathcal{O}_c$.

Let j^* be the order in $\mathcal{O}_c$ such that $W(\mathcal{O}_c)_{j^*} \geq W(\mathcal{O}_c)_k$ for all k in $\mathcal{O}_c$.

Update $CL_i = CL_i - p_{ij^*}$ for $1 \leq i \leq m$.

Delete j^* from π_e, and add j^* to π_l.

Step 4: {*Stopping Criterion*}

If $j = n$, then STOP;

Otherwise increase j by 1 and go to Step 2.

Clearly, step 1 takes $O(n \lg n + nm)$ for initialization. Each iteration of both step 2 and step 3 takes $O(nm)$ time. The algorithm runs at most n iterations of step 2 and step 3. Thus, the overall running time of the algorithm is $O(n^2m)$.

In implementation, the heuristic is run with 3 different pairs of ω_1 and ω_2: (i) $\omega_1 = 1$ and $\omega_2 = 0$; (ii) $\omega_1 = 0$ and $\omega_2 = 1$; (iii) $\omega_1 = 1$ and $\omega_2 = 1$. Each setting of the weights generates a schedule and the best schedule is kept.

3.3.3 An Exact Algorithm for $PD \mid\mid \sum U_j$

For convenience of description, the following notation is defined:

$\mathcal{S}$: A partial schedule $(j_1, j_2, \ldots, j_k), 1 \leq k \leq n$, that has already been established for the first k positions. It consists of a set of orders that are completed by their due dates.

$\mathcal{L}$: The set of late orders so far.

$\mathcal{O}'$: The set of orders already scheduled. Clearly, $\mathcal{O}' = \mathcal{S} \cup \mathcal{L}$.

$\mathcal{O}''$: The set of orders still to be scheduled.

$\sum U^*_{ij}(r_i, \mathcal{O}'')$: The minimum $\sum U_j$ obtained after scheduling the orders in $\mathcal{O}''$ on machine i; the minimum is obtained by considering machine i as a separate single machine problem independent from the other machines and assuming it is available from time r_i on.

$\sum U^*_j(\mathcal{S}, \mathcal{O}'')$: The minimum $\sum U_j$ given the initial partial schedule $\mathcal{S}$ and assuming that the schedule of the orders in $\mathcal{O}''$ is optimized. In other words, $\sum U^*_j(\mathcal{S}, \mathcal{O}'')$ is the best objective value of all possible schedules that start with $\mathcal{S}$.

To design an efficient exact algorithm for $PD \mid\mid \sum U_j$, elimination criteria are important to make the algorithm run faster. For example, it is easy to see that there exists an optimal schedule with all non-tardy orders scheduled first in increasing order of their due dates, followed by all tardy orders in any sequence. Thus, schedules violating this ordering can be eliminated. Also, order dominance serves as an elimination criterion. Basically, these criteria help identify orders that can be eliminated from further consideration, based on a partial schedule $\mathcal{S}$. However, more can be done. One can apply bounding techniques on $\mathcal{S}$, to see if it is possible to obtain better solutions starting out from $\mathcal{S}$. If not, then all schedules starting out from $\mathcal{S}$ can be discarded. The following lemma presents a lower bound on the number of late orders based on $\mathcal{S}$.

Lemma 3.10 (Lower Bound on Number of Late Orders) *If $\mathcal{S}$ is the current partial schedule, $\mathcal{L}$ the set of late orders so far, and $\mathcal{O}''$ the set of orders still to be scheduled, then*

$$\sum U_j^*(\mathcal{S}, \mathcal{O}'') \geq |\mathcal{L}| + \max_{1 \leq i \leq m} \left\{ \sum U_{ij}^*(C_{ij_k}, \mathcal{O}'') \right\},$$

where C_{ij_k} is the finish time of the last non-tardy order j_k on machine i.

Proof: Consider an optimal schedule $\mathcal{S}_b$ started with the $\mathcal{S}$, note that both $\mathcal{S}_b$ and $\mathcal{S}$ consist of a sequence of non-tardy orders followed by a set of tardy orders in any sequence. It is clear that $\mathcal{S}$ is a part of $\mathcal{S}_b$. Let $\mathcal{S}_c$ denote the sub-sequence of non-tardy orders scheduled behind $\mathcal{S}$ for the orders in $\mathcal{O}''$. Consider $\mathcal{S}_c$ on machine i $(1 \leq i \leq m)$. Clearly, $\mathcal{S}_c$ starts on machine i at time C_{ij_k}. It would never be better than the optimal schedule of $\mathcal{O}''$ on machine i starting at time C_{ij_k}, which is denoted by $\sum U_{ij}^*(C_{ij_k}, \mathcal{O}'')$. That is,

$$\sum U_j^*(\mathcal{S}_c) \geq \sum U_{ij}^*(C_{ij_k}, \mathcal{O}''), \quad i = 1, \ldots, m.$$

Therefore,

$$\begin{aligned}\sum U_j^*(\mathcal{S},\mathcal{O}'') &= |\mathcal{L}|+\sum U_j(\mathcal{S}_c)\\ &\geq |\mathcal{L}|+\max_{1\leq i\leq m}\{\sum U_{ij}^*(C_{ij_k},\mathcal{O}'')\}.\end{aligned}$$

The result follows. □

Now, it is ready to present an exact algorithm that systematically searches for an optimal schedule, by using the elimination criteria and the lower bound. The exact algorithm combines the Check-Forward algorithm with bounding technique. The Check-Forward algorithm, which is based on constraint propagation and backtracking, has been widely applied to solve Constraint Satisfaction Problems in various fields like Artificial Intelligence and Operations Research. For a survey of algorithms (including the Check-Forward algorithm) for Constraint Satisfaction Problems, the reader is referred to Kumar [36].

In the check-forward phase of the algorithm, it maintains a sequence of variables $o_1, o_2, \ldots, o_n$ to be instantiated one by one with candidate orders from the domains of these variables. The initial domain for any position o_ℓ is the set of all orders to be scheduled. For a partial schedule $o_1, \ldots, o_k$, the sets of candidate orders for each one of the remaining positions $o_{k+1}, \ldots, o_n$ can be found through constraint propagation starting out from the current partial schedule $o_1, \ldots, o_k$ (in which a total of k orders have already been fixed with all k orders completed by their due dates) and using the elimination criteria mentioned before. While fixing more and more positions, the domain sizes of the remaining positions become smaller and smaller. A terminal schedule is reached at position k and a partial schedule $o_1, \ldots, o_k$ is generated without any late orders and the domain of position o_{k+1} is empty. Therefore, the number of late orders in such a terminal schedule is $(n-k)$. In order to search for an optimal schedule with a minimum number of tardy orders, a backtracking procedure is needed to try all other possible positions of orders.

To further accelerate the search procedure, a bounding strategy, which is an essential part of the Branch and Bound algorithm, is used. It is clear that the initial upper bound UB for the number of late orders can be generated by the *GHM* heuristic presented previously. Lower bounds are established for a partial schedule $o_1, \dots, o_k$ in order to eliminate those schedules starting out from $o_1, \dots, o_k$ but with objective values no less than UB. Let $\mathcal{J}'$ denote the set of unscheduled orders after fixing o_k. In order to calculate a lower bound LB on the minimum number of late orders for all schedules starting out with the partial schedule $o_1, \dots, o_k$, the Hodgson-Moore algorithm is applied to solve a $1 \mid r_j = C_{o_k} \mid \sum U_j$ problem for the orders in $\mathcal{J}'$ on each of the m machines, where C_{i,o_k} is the finish time of the order scheduled in position o_k on machine i. Let l_i be the number of late orders for scheduling the orders in $\mathcal{J}'$ on machine i. Then by Lemma 3.10,

$$LB = (n - |\mathcal{J}'| - k) + \max_{1 \le i \le m} \{l_i\},$$

where $(n - |\mathcal{J}'| - k)$ is the number of late orders already produced to get the partial schedule $o_1, \dots, o_k$. Clearly, if $UB \le LB$, then no better solutions can be found with the current partial schedule $o_1, \dots, o_k$. Therefore, o_k should be assigned a different order from its domain of candidate orders. After all possible orders in the candidate set of o_k have been checked out, the procedure backtracks to position o_{k-1}. The procedure continues in this fashion until all orders in the candidate set of o_1 are exhausted. During the search procedure, UB is updated whenever a smaller number of late orders is found.

The above idea is implemented as the following *Exact Algorithm* which combines constraint propagation, backtracking, and bounding techniques. Without loss of generality it is assumed that $d_1 \le d_2 \le \cdots \le d_n$.

The Exact Algorithm

Step 1: {*Initialization*}

Let UB be the number of late orders obtained through the *GHM* heuristic.

Let $J_{o_1} = \{1, 2, \dots, n\}$ be the domain of candidate order for position o_1.

Let $k = 1$.

Let $\sigma^*[1..n]$ be a sequence to maintain the best schedule.

Step 2: {*Constraint Propagation and Searching*}

Choose the first order j from J_{o_k}. If no order exists, goto Step 5.

Otherwise, let $o_k = j$ and $J_{o_k} = J_{o_k} \setminus \{j\}$.

Resolve J_{o_k} by dominance conditions.

Let C_{i,o_k} be the finish time of order $o_k = j$ on machine $i, i = 1, 2, \dots, m$.

If order $o_k = j$ is completed after its due date, then repeat Step 2.

Step 3: {*Adjustment of Upper Bound on the Number of Late Orders*}

Otherwise, if $n - k < UB$ then $UB = n - k$ and $\sigma^*[1..k] = o_1, \dots, o_k$.

If $k = n$ goto Step 6.

Step 4: {*Establishing Lower Bound on the Number of Late Orders*}

Solve for each machine $i = 1, 2, \dots, m$, a $1 \mid r_j = C_{i,o_k} \mid \sum U_j$ problem by applying the Hodgson-Moore algorithm to the set of orders J_{o_k}. {Lemma 3.10}

Let the number of late orders on machine i be l_i.

Let $LB = (n - |J_{o_k}| - k) + \max_{1 \le i \le m}\{l_i\}$. {Bounding}

If $UB > LB$ then $J_{o_{k+1}} = J_{o_k}$, increase k by 1. {Expanding}

Go to Step 2.

Step 5: {*Standard Stopping Criterion*}

If $k = 1$, then assign late orders $\{1, 2, \ldots, n\} \setminus \{\sigma^*[1..(n - UB)]\}$ in arbitrary order to $o_{n-UB+1}, o_{n-UB+2}, \ldots, o_n$, and return $\sigma^*[1..n]$ as an optimal solution with UB as the number of late orders.

Otherwise, decrease k by 1, goto Step 2. {Backtracking}

Step 6: {*Shortcut Stopping Criterion*}

Return $\sigma^*[1..n]$ as an optimal solution with 0 as the number of late orders.

Clearly, the exact algorithm takes an exponential amount of time in the worst case. However, introducing constraint propagation and bounding techniques in the algorithm is expected to reduce the running time to a large extent.

3.4 Empirical Analysis

To test the algorithms, problem instances of various sizes are generated. The sizes of the problem instances are determined by the number of orders n and the number of machines m, where $n = 20, 50, 100, 200$ and $m = 2, 5, 10, 20$. For each combination of n and m, problem instances of varying hardness are generated according to different characteristics of the due dates.

For each order $j, 1 \leq j \leq n$, the number of product types l_j is first randomly generated from the uniform distribution $[1, m]$, then the l_j machines are randomly chosen from the machine set. For each machine i that is selected in this manner, an integer processing time p_{ij} is generated from the uniform distribution $[1, 100]$.

Finally, the due dates of the orders are generated based on the setting of two parameters Δ_1 and Δ_2. For each order j, an integer due date d_j is randomly generated from the uniform distribution

$$[P(1 - \Delta_1/2 - \Delta_2), P(1 + \Delta_1/2 - \Delta_2)],$$

Table 3.1 Optimality Percentage and Average CPU Time (in Seconds) vs. Each Combination of n and m

	$m = 2$		$m = 5$		$m = 10$		$m = 20$	
n	%	CPU avg	%	CPU avg	%	CPU avg	%	CPU avg
20	100	0.001	100	0.002	100	0.003	100	0.007
50	100	9	100	22	100	68	100	160
100	76	37	84	504	84	775	76	527
200	48	749	60	686	56	1299	56	99
	avg(%)=84%							

where

$$P = \sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}/m.$$

(In order to avoid negative due dates, it is assumed that $d_j \geq 0$, i.e., if $d_j < 0$, then it is forced to be 0.) The rationale for generating the due dates in this manner is the following: The P gives an indication of the average time each machine is occupied. The value of Δ_1 determines the range in which the due dates lie and the value of Δ_2 is used for adjusting the tightness of the due dates.

The values of Δ_1 and Δ_2 are chosen from the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. For each combination of Δ_1 and Δ_2, an instance is generated for each pair of n and m. In total, 400 instances are generated.

The algorithms are implemented in C++ with STL. The running environment is the RedHat Linux 7.0 operating system running on a Pentium II (400Mhz) PC with 128MB RAM. In the experiment, the time limit for the exact algorithm is set to 7 days. For detailed experimental results produced by the algorithms, the reader is referred to Leung, Li, and Pinedo [46]. In what follows, some findings are listed based on the statistical data of the experimental results.

Table 3.2 Optimality Percentage vs. Each Setting of Δ_1 and Δ_2 for $n = 100, 200$

	Δ_1					Δ_2				
	0.2	0.4	0.6	0.8	1.0	0.2	0.4	0.6	0.8	1.0
% solved	35	42.5	65	92.5	100	97.5	75	42.5	47.5	77.5

Table 3.1 shows the percentage of all instances that are solved to optimality (%) by the exact algorithm together with the average CPU time in seconds (CPU avg), for each combination of n and m. In average, 84% of the generated instances can be solved to optimality. In particular, all instances of $n = 20$ and 50 are solved to optimality very quickly.

Table 3.2 shows the percentage of instances solved to optimality (% solved) versus each setting of Δ_1 and Δ_2 for $n = 100, 200$. Note that instances for $n = 20, 50$ are not counted here, since all these instances are solved to optimality. The results in this table reveals that smaller Δ_1 implies harder problem instances for the exact algorithm. The table also shows that the exact algorithm performs well for $\Delta_2 = 0.2, 0.4, 1.0$, but not for $\Delta_2 = 0.6, 0.8$. Since Δ_2 determines the tightness of the due dates, it affects the expected number of late orders. For $\Delta_2 = 0.6, 0.8$, the expected number of late orders would be about half of the number of orders. Thus, more expanding and backtracking iterations are needed for such instances. However, for $\Delta_2 = 0.2, 0.4, 1.0$, the bounding strategy might help much to reduce either expanding or backtracking iterations. Thus, the algorithm performs better on these instances.

For those problem instances that are solved by the exact algorithm, Table 3.3 shows the percentage of these instances that are also returned optimal solutions by the *GHM* heuristic. The heuristic achieves an optimal solution for about 60% of the instances that can be solved with the exact algorithm.

Table 3.4 shows the average improvement per instance by the exact algorithm for each combination of n and m. It reveals that the results obtained by the *GHM*

Table 3.3 Optimality Percentage of the *GHM* Heuristic vs. Each Combination of n and m

	m			
n	2	5	10	20
20	88%	88%	92%	28%
50	64%	64%	44%	48%
100	53%	48%	43%	47%
200	83%	53%	57%	64%
	avg=60.25%			

Table 3.4 Average Improvement per Instance Obtained by the Exact Algorithm from *GHM* for Each Combination of n and m

	m			
n	2	5	10	20
20	0.12	0.16	0.05	0.4
50	0.44	0.48	0.88	0.68
100	0.64	0.76	0.92	0.88
200	0.24	0.6	0.68	0.44
	avg=0.52			

heuristic is actually very close to those obtained by the exact algorithm. In average, the improvement achieved by the exact algorithm is about 0.52 per instance. This implies that the *GHM* heuristic performs quite well in practice.

3.5 Concluding Remarks

In this chapter, the fully dedicated case with some due date related objectives was investigated. Special interest was focused on the minimization of the maximum lateness $L_{\max}$ and the total number of late orders $\sum U_j$. Only the most basic scheduling problems have been considered. There are many more general scheduling problems with due date related objectives that deserve attention.

For example, an interesting problem to study with the $L_{\max}$ objective would be $PD \mid r_j \mid L_{\max}$. It is clear that this problem is strongly NP-hard. However, the single machine version already has received a fair amount of attention in the literature.

With regard to the $\sum U_j$ objective many more general problems appear to be of interest. The heuristic presented in this chapter appears to work well for $PD \mid\mid \sum U_j$. It may be of interest to adapt this heuristic to the following more general problems:

(i) $PD \mid r_j, prmp \mid \sum U_j$,

(ii) $PD \mid r_j \mid \sum U_j$,

(iii) $PD \mid\mid \sum w_j U_j$,

(iv) $PD \mid r_j, prmp \mid \sum w_j U_j$, and

(v) $PD \mid r_j \mid \sum w_j U_j$.

It is clear that release dates can make a nonpreemptive problem considerably harder. However, when with release dates preemptions are allowed, it still may be possible to design very effective heuristics. These heuristics may be somewhat more complicated than the one described in this chapter, but may be equally effective. The problem without release dates and with weights may give rise to different types of heuristics.

Nonpreemptive problems with release dates and weights are, of course, very hard. The hardest problem among those mentioned above most likely is $PD \mid r_j \mid \sum w_j U_j$.

CHAPTER 4

THE FULLY DEDICATED CASE – THE TOTAL WEIGHTED COMPLETION TIME

4.1 Introduction

In Chapter 2 and Chapter 3, some results were established for the fully dedicated case with unweighted objectives. This chapter still examines the fully dedicated case. However, the objectives to be optimized are now weighted functions. Special attention is focused on the minimization of the total weighted completion time. The two problems of interest are $PD \mid\mid \sum w_j C_j$ and $PD \mid r_j \mid \sum w_j C_j$.

While $PD1 \mid\mid \sum w_j C_j$ can be solved by the Smith's ratio rule (see [62]), the problem $PDm \mid\mid \sum w_j C_j$ has been shown to be strongly NP-hard for any fixed number (≥ 2) of machines (see Sung and Yoon [63]). The more general problem $PDm \mid r_j \mid \sum w_j C_j$ is strongly NP-hard even for $m = 1$ (see Lenstra et al. [42]).

Due to the NP-hardness of the problems, this chapter focuses on the performance analyses of a number of priority rules and approximation algorithms for $PD \mid\mid \sum w_j C_j$ and $PD \mid r_j \mid \sum w_j C_j$. The algorithms considered are of two types: priority rules (either static or dynamic) and LP-based algorithms. The priority rules are only applicable to $PD \mid\mid \sum w_j C_j$, i.e., when $r_j = 0$ for all j. An analysis of the priority rules shows that they are sensitive to the characteristics of processing times of the orders. It appears that the more sophisticated LP-based algorithms are not so sensitive to the characteristics of the processing times. In order to obtain some insights into the performance of these algorithms when applied to instances of $PD \mid\mid \sum w_j C_j$ in practice, this chapter also presents an extensive comparative analysis considering solution quality, speed, memory space, and implementation complexity.

The observations from the empirical analysis may be very helpful for the selection of an appropriate rule or algorithm in a real life situation.

The results in this chapter also appear in Leung, Li, and Pinedo [47] which has been submitted for publication. The chapter is organized as follows. Section 4.2 presents five priority rules and analyzes their performance bounds. Section 4.3 studies the performance bounds of these rules assuming additional constraints on the characteristics of the processing times of each order. Section 4.4 focuses on two LP-based approximation algorithms and Section 4.5 presents an empirical analysis of the rules and algorithms. Finally, Section 4.6 contains some concluding remarks.

4.2 Priority Rules for $PD \mid\mid \sum w_j C_j$

Throughout this section it is assumed that $r_j = 0$ for all j. In addition, preemptions are not allowed and only permutation schedules are considered.

Let Ω denote the set of unscheduled orders; let π denote a partial schedule. Five greedy ways are considered for selecting the next order $j^* \in \Omega$ to be added to the partial schedule. The first two methods described below are basically static priority rules, i.e., the entire sequence can be determined at time $t = 0$ based only on information pertaining to the orders. The third method is a two-pass rule, which schedules the orders in two passes (the schedule information obtained in the first pass provides the data necessary for doing the second pass). The fourth and fifth method are single pass dynamic priority rules. That is, the schedule can be developed in a single pass; however, it cannot be done using only information pertaining to the orders. In order to add an additional order to a partial schedule, information pertaining to the existing partial schedule has to be taken into account as well. The fifth method can also be enhanced with a limited postprocessing procedure.

Definition 4.1 *The Weighted Shortest Total Processing Time first (WSTP) rule schedules the orders in increasing order of* $\sum_{i=1}^{m} p_{ij}/w_j$.

Definition 4.2 *The Weighted Shortest Maximum Processing Time first (WSMP) rule schedules the orders in increasing order of* $\max_i\{p_{ij}\}/w_j$.

Definition 4.3 *The Weighted Smallest Maximum Completion Time first (WSMC) rule first sequences the orders on each machine* $i = 1, 2, \ldots, m$, *in increasing order of* p_{ij}/w_j. *(Note that the order sequences on the various machines may be different.) The rule then computes the completion time for order* j *as* $C_j' = \max_{i=1}^{m}\{C_{ij}\}$. *In its second pass, the rule schedules the orders in increasing order of* C_j'.

Definition 4.4 *The Weighted Shortest Processing Time first applied to the machine with the largest load (WSPL) functions as a dynamic priority rule that generates a sequence of orders one at a time, each time selecting as the next order the order* $j^* \in \Omega$ *such that* $j^* = \arg\min_{j\in\Omega}\left\{\frac{p_{i^*j}}{w_j}\right\}$ *where* i^* *is the machine with the largest workload under the partial schedule* π.

Definition 4.5 *The Weighted Earliest Completion Time first (WECT) rule selects as the next order* j^* *which satisfies* $j^* = \arg\min_{j\in\Omega}\{(C_j - C_k)/w_j\}$, *where* C_k *is the finish time of the order scheduled immediately before order* j^*. *Ties may be broken arbitrarily. After an order* j^* *has been selected by* $WECT$ *to be included in the partial schedule, a postprocessing procedure that will be described shortly may be invoked.*

The postprocessing procedure invoked by $WECT$ interchanges order j^* with order k in case $C_{j^*} \leq C_k$ in order to obtain a better (or at least not a worse) solution. Note that the case $C_{j^*} \leq C_k$ occurs only when $p_{i^*j^*} = 0$, where i^* is the machine on which order k has, over all machines, the largest finish time. Assume that after the swap the order immediately before order j^* is order l. If $C_{j^*} \leq C_l$, then it proceeds with an interchange of order j^* with order l. This postprocessing procedure is repeated until C_{j^*} is larger than the completion time of the order that immediately precedes it. Note that after each swap, the finish time of j^* either decreases or remains unchanged, while the finish time of each order that is swapped with j^* remains unchanged. This is due to the fact that order j^* has zero processing time on the machine on which the swapped order has its largest finish time. Thus, the postprocessing, if any, produces a solution that is no worse than the one without postprocessing.

Note that when applying the $WECT$ rule, there may at times be ties. Since ties may be broken arbitrarily, the $WECT$ rule could lead to various different schedules with different values of objective functions.

Through a sorting algorithm, both $WSTP$ and $WSMP$ can be implemented to run in $O(mn + n \lg n)$ time, and $WSMC$ can be implemented to run in $O(mn \lg n)$ time. Both $WSPT$ and $WECT$ can be implemented in a rather straightforward manner to run in $O(mn^2)$ time.

Now consider the performance bounds of the five rules. When $m = 2$, Sung and Yoon [63] showed that both $WSTP$ and $WSMC$ have an approximation ratio of 2. Actually, Wang and Cheng [66] obtained the following result:

Theorem 4.1 *$WSTP$, $WSMP$ and $WSMC$ are all m-approximation algorithms for $PD \mid\mid \sum w_j C_j$.*

As for $WSPL$, it has been shown in Chapter 2 that the algorithm is unbounded even when all $w_j = 1$. However, an empirical analysis showed that it performs very well in practice for $w_j = 1$. The $WECT$ algorithm is a certain generalization of the m-approximation ECT algorithm introduced in Chapter 2. If all $w_j = 1$, then $WECT$ reduces to ECT. In what follows, it will be shown that $WECT$ is also an m-approximation algorithm for $PD \mid\mid \sum w_j C_j$.

Let $p_j = \max(p_{1j}, \ldots, p_{mj})$, $j = 1, 2, \ldots, n$. Let $C_j(WECT)$ and $C_j(OPT)$ denote the completion time of order j in the $WECT$ schedule and the optimal schedule, respectively. Assume without loss of generality that the orders are labeled in such a way that

$$\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_n}{w_n}. \tag{4.1}$$

Furthermore, let $[j]$ refer to the order put in position j of a schedule.

First, the following lemma is a key observation with regard to $WECT$:

Lemma 4.2 *For any schedule generated by the $WECT$ rule without postprocessing,*

$$\frac{C_{[j]}(WECT) - C_{[j-1]}(WECT)}{w_{[j]}} \leq \frac{p_j}{w_j}, j = 2, 3, \ldots, n.$$

Proof: The proof is by contradiction. According to the definition of $WECT$, the order selected for the first position is order 1. Thus,

$$C_{[1]}(WECT) = p_1, \text{ and } w_{[1]} = w_1.$$

Suppose that there exists a smallest position j^* $(1 < j^* < n)$ of the orders scheduled according to $WECT$, such that

$$\frac{C_{[j^*]}(WECT) - C_{[j^*-1]}(WECT)}{w_{[j^*]}} > \frac{p_{j^*}}{w_{j^*}}.$$

For any position j, it is easy to see that

$$C_{[j]}(WECT) - C_{[j-1]}(WECT) \leq p_{[j]}, \quad j = 2, 3, \ldots, n. \tag{4.2}$$

Therefore, if the order scheduled in position j^* is order j^* itself, then it satisfies

$$\frac{C_{[j^*]}(WECT) - C_{[j^*-1]}(WECT)}{w_{[j^*]}} = \frac{C_{j^*}(WECT) - C_{[j^*-1]}(WECT)}{w_{j^*}} \leq \frac{p_{j^*}}{w_{j^*}}.$$

The reason why the $WECT$ rule did not put order j^* in this position must be that it had already been selected earlier. However, this immediately implies that there exists at least one order j' $(2 \leq j' \leq j^* - 1)$ that has not yet been selected by the algorithm. Now, if order j' is put in position j^*, then it satisfies

$$\frac{C_{[j^*]}(WECT) - C_{[j^*-1]}(WECT)}{w_{[j^*]}} = \frac{C_{j'}(WECT) - C_{[j^*-1]}(WECT)}{w_{j'}} \leq \frac{p_{j'}}{w_{j'}} \leq \frac{p_{j^*}}{w_{j^*}},$$

due to the assumption of the ordering in (4.1). This leads to a contradiction, since $WECT$ always chooses as the next order the one with the smallest value of

$$\frac{C_{[j^*]}(WECT) - C_{[j^*-1]}(WECT)}{w_{[j^*]}}.$$

The result follows. □

Based on Lemma 4.2, the following upper bound can be established for the objective costs of schedules produced by $WECT$.

Lemma 4.3 *For any schedule generated by the $WECT$ rule without postprocessing,*

$$\sum_{j=1}^{n} w_{[j]} C_{[j]}(WECT) \leq \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k.$$

Proof: Let $p'_{[1]} = p_1$, $w_{[1]} = w_1$ and let

$$p'_{[j]} = C_{[j]}(WECT) - C_{[j-1]}(WECT), \quad j = 2, 3, \ldots, n.$$

Note that, by Lemma 4.2,

$$\frac{p'_{[j]}}{w_{[j]}} \leq \frac{p_j}{w_j}, \quad j = 2, 3, \ldots, n; \tag{4.3}$$

and

$$p'_{[j]} \leq p_{[j]}, \quad j = 2, 3, \ldots, n. \tag{4.4}$$

It is clear that each $p'_{[j]}$ can be easily determined from the $WECT$ schedule. Based on this notation,

$$\sum_{j=1}^{n} w_{[j]} C_{[j]}(WECT) = \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} p'_{[k]}. \tag{4.5}$$

In order to prove the lemma, it suffices to show that

$$\sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} p'_{[k]} \leq \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.6}$$

To prove (4.6), it would be helpful to consider an artificial instance of the classical problem $1 \; || \; \sum w_j C_j$, with jobs whose processing times and weights are given by the following pairs:

$$(p_1, w_1), (p_2, w_2), \ldots, (p_n, w_n).$$

Here, each pair of p_j and w_j are exactly the same as those in (4.1). Clearly, if the above jobs are sequenced according to the classical *WSPT* rule, then the objective of the schedule is

$$\sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k,$$

due to the assumption of the ordering in (4.1). For convenience, this very first sequence is referred to as

$$S_1 : \; < 1, 2, \ldots, n > . \tag{4.7}$$

Now change in S_1 the processing time of the job, whose label is the same as that of the order scheduled in the second position of *WECT*. Note that this order is labeled as [2] in *WECT* and it corresponds to one specific job scheduled in S_1. For convenience and consistency, the label of this job in S_1 is referred to as [2]. Since the *WECT* algorithm always chooses order 1 as [1], it is clear that $[2] \in \{2, 3, \ldots, n\}$. With [2], S_1 can be rewritten as:

$$S_1 : \; < 1, 2, \ldots, [2]-1, [2], [2]+1, \ldots, n > .$$

Now, for job [2] in S_1, let

$$p_{[2]} = p'_{[2]}.$$

According to (4.4), $p_{[2]}$ becomes equal to or smaller than its initial value, while $w_{[2]}$ remains unchanged. It follows that the objective cost of S_1 remains unchanged or decreases after letting $p_{[2]} = p'_{[2]}$, i.e.,

$$\sum w_j C_j(S_1) \leq \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.8}$$

Note that if [2] = 2, the above inequality still holds and this case is simple. Thus, only the case $[2] \neq 2$ needs to be focused on. After letting $p_{[2]} = p'_{[2]}$, from (4.1) and (4.3) one can obtain a new ordering of p_j/w_j as follows:

$$\frac{p_1}{w_1} \leq \frac{p_{[2]}}{w_{[2]}} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_{[2]-1}}{w_{[2]-1}} \leq \frac{p_{[2]+1}}{w_{[2]+1}} \leq \ldots \leq \frac{p_n}{w_n}. \tag{4.9}$$

Swapping in S_1 the position of job [2] with that of job $[2]-1$ results in the following new sequence:

$$S_2 : \; < 1, 2, \ldots, [2]-2, [2], [2]-1, [2]+1, \ldots, n > . \tag{4.10}$$

According to (4.9), since

$$\frac{p_{[2]}}{w_{[2]}} \leq \ldots \leq \frac{p_{[2]-1}}{w_{[2]-1}},$$

it is easy to show via an adjacent interchange argument (Pinedo [57]) that

$$\sum w_j C_j(S_2) \leq \sum w_j C_j(S_1). \tag{4.11}$$

Repeatedly swap the position of job [2] with that of the job positioned immediately before it until the following sequence is obtained:

$$S_3 : \; < 1, [2], 2, \ldots, [2]-1, [2]+1, \ldots, n >, \tag{4.12}$$

and

$$\sum w_j C_j(S_3) \leq \sum w_j C_j(S_2). \tag{4.13}$$

The above procedure is repeated for each job $[3], [4], \ldots, [n]$ corresponding to the labels of the orders in $WECT$, it finally produces the following sequence:

$$S_4 : \ < 1, [2], [3], \ldots, [n-1], [n] >, \tag{4.14}$$

and

$$\sum w_j C_j(S_4) \leq \sum w_j C_j(S_3). \tag{4.15}$$

From (4.8),(4.11),(4.13) and (4.15), it is easy to see that

$$\sum w_j C_j(S_4) \leq \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.16}$$

Note that in S_4, for each $j = 2, 3, \ldots, n$

$$p_{[j]} = p'_{[j]}.$$

It follows that

$$\sum w_j C_j(S_4) = \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p'_{[k]}. \tag{4.17}$$

From (4.5), (4.16) and (4.17), the result follows. □

Theorem 4.4 *For $PD \mid\mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(WECT)}{\sum w_j C_j(OPT)} \leq m.$$

Proof: Even without the postprocessing procedure, Lemma 4.3 shows that

$$\sum w_j C_j(WECT) \leq \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.18}$$

For the optimal schedule,

$$C_{[j]}(OPT) = \max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{i[k]}\right\} \ge \sum_{k=1}^{j}\left(\sum_{i=1}^{m} p_{i[k]}\right)/m \ge \sum_{k=1}^{j}\max_{1\le i\le m}\{p_{i[k]}\}/m.$$

It follows that

$$\sum_{j=1}^{n} C_j(OPT) \ge \sum_{j=1}^{n} w_{[j]}\sum_{k=1}^{j}\max_{1\le i\le m}\{p_{i[k]}\}/m \ge \sum_{j=1}^{n} w_j\sum_{k=1}^{j} p_k/m. \tag{4.19}$$

Note that the last "$\ge$" in (4.19) is due to Smith's $WSPT$ rule. Hence, from (4.18) and (4.19), the performance ratio of $WECT$ is at most m even without postprocessing. Since the postprocessing could lead to a better solution, it helps to improve the performance of $WECT$. □

4.3 Analyses of Priority Rules with Constraints on Processing Times

It would not be surprising that the priority rules may perform better when the processing times of each order are subject to constraints that ensure some form of regularity in the processing times. Sung and Yoon [63] showed that for $m = 2$ the performance ratio of $WSTP$ can be reduced to 3/2 when the processing times satisfy the additional constraint $(p_{1j} + p_{2j})/2 \ge |p_{1j} - p_{2j}|$ for each $j = 1, 2, \ldots, n$. In what follows, tighter bounds will be shown for the rules when the processing times are subject to additional constraints. In the remaining part of this section, the following notation is used:

- For each $j = 1, 2, \ldots, n$ let $\delta_j = \max_{1\le i\le m}\{p_{ij}\} - \min_{1\le i\le m}\{p_{ij}\}$.
- Let $\min^{(k)}_{1\le i\le m}\{a_i\}$ denote the k^{th} smallest item among $\{a_1, a_2, \ldots, a_m\}$.
- Let $[j]$ denote the order scheduled in position j of a schedule.

4.3.1 Additional Constraints $\sum_{i=1}^{m} p_{ij}/m \ge \delta_j$

With the additional constraints $\sum_{i=1}^{m} p_{ij}/m \ge \delta_j$ for each order $j = 1, 2, \ldots, n$, the following result can be shown:

Theorem 4.5 *For $PD \mid\mid \sum w_jC_j$, if*

$$\sum_{i=1}^{m} p_{ij}/m \geq \delta_j$$

for each order $j = 1, 2, \ldots, n$, then

$$\frac{\sum w_jC_j(WSTP)}{\sum w_jC_j(OPT)} \leq 2 - \frac{1}{m}.$$

Proof: Without loss of generality, it may be assumed that

$$\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_n}{w_n}, \tag{4.20}$$

where $p_j = \sum_{i=1}^{m} p_{ij}$, $j = 1, 2, \ldots, n$. According to such notation,

$$\delta_j \leq p_j/m. \tag{4.21}$$

It is clear that

$$\sum w_jC_j(OPT) \geq \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} p_{[k]}/m \geq \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k/m. \tag{4.22}$$

Again, the last "$\geq$" in (4.22) is due to Smith's $WSPT$ rule.

Now consider the schedule generated by $WSTP$. Clearly, in this schedule, the order scheduled in position j is order j itself (If there are ties, one can always relabel the orders such that they satisfy the ordering in (4.20)). Suppose the latest finish time of order j takes place on machine i^*, and the earliest finish time of order j is on machine i'. Now

$$\max_{1\leq i\leq m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \min_{1\leq i\leq m}\left\{\sum_{k=1}^{j} p_{ik}\right\} = \sum_{k=1}^{j} p_{i^*k} - \sum_{k=1}^{j} p_{i'k} = \sum_{k=1}^{j}(p_{i^*k} - p_{i'k}) \leq \sum_{k=1}^{j} \delta_k. \tag{4.23}$$

Therefore, for each $i = 1, 2, \ldots, m$ the following relation holds:

$$\max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \overset{(i)}{\min_{1\le l\le m}}\left\{\sum_{k=1}^{j} p_{lk}\right\} \le \max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \min_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{ik}\right\} = \sum_{k=1}^{j} \delta_k. \tag{4.24}$$

Or, equivalently,

$$\overset{(i)}{\min_{1\le l\le m}}\left\{\sum_{k=1}^{j} p_{lk}\right\} \ge \max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \sum_{k=1}^{j} \delta_k. \tag{4.25}$$

Thus,

$$\begin{aligned} C_j(WSTP) &= \max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{ik}\right\} = \sum_{k=1}^{j} p_k - \sum_{i=1}^{m-1} \overset{(i)}{\min_{1\le l\le m}}\left\{\sum_{k=1}^{j} p_{lk}\right\} \\ &\le \sum_{k=1}^{j} p_k - (m-1)\left(\max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \sum_{k=1}^{j} \delta_k\right). \end{aligned} \tag{4.26}$$

Or, equivalently,

$$C_j(WSTP) = \max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{ik}\right\} \le \frac{\sum_{k=1}^{j} p_k + (m-1)\sum_{k=1}^{j} \delta_k}{m}.$$

Therefore, by (4.21),

$$C_j(WSTP) \le \sum_{k=1}^{j} \frac{(2m-1)p_k}{m^2}. \tag{4.27}$$

From (4.22) and (4.27), the result follows. □

It is clear that the above bound is monotonically increasing with m. It is 1 when $m = 1$, and 3/2 when $m = 2$. When m becomes infinitely large, the bound is 2.

Theorem 4.6 *For $PD \,||\, \sum w_j C_j$, if*

$$\sum_{i=1}^{m} p_{ij}/m \ge \delta_j$$

for each order $j = 1, 2, \ldots, n$, then

$$\frac{\sum w_j C_j(WSMP)}{\sum w_j C_j(OPT)} \le \frac{1}{1 + \mathcal{H}(m) - \mathcal{H}(2m-1)},$$

where $\mathcal{H}(k) \equiv 1 + \frac{1}{2} + \ldots + \frac{1}{k}$ *is the harmonic series.*

Proof: Without loss of generality it may be assumed that

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \ldots \le \frac{p_n}{w_n}, \tag{4.28}$$

where $p_j = \max_{1\le i\le m}\{p_{ij}\}$, $j = 1, 2, \ldots, n$.

In the schedule generated by $WSMP$, the order scheduled in position j is order j itself (Again, if there are ties, one can always relabel the orders to guarantee the ordering in (4.28)). It is easy to see that

$$\sum w_j C_j(WSMP) \le \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.29}$$

Now consider the order scheduled in the j^{th} position of an optimal solution. Its completion time is

$$C_{[j]}(OPT) = \max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{i[k]}\right\} \ge \frac{\sum_{k=1}^{j}\sum_{i=1}^{m} p_{i[k]}}{m}. \tag{4.30}$$

According to the assumption, for the order scheduled in the k^{th} position

$$\frac{(m-1)\max_{1\le i\le m}\{p_{i[k]}\} + \min_{1\le i\le m}\{p_{i[k]}\}}{m} \ge \frac{\sum_{i=1}^{m} p_{i[k]}}{m} \ge \max_{1\le i\le m}\{p_{i[k]}\} - \min_{1\le i\le m}\{p_{i[k]}\}.$$

It follows that

$$\min_{1\le i\le m}\{p_{i[k]}\} \ge \frac{p_{[k]}}{m+1}. \tag{4.31}$$

More generally, for each $i = 1, 2, \ldots, m$

$$\frac{(m-i)\max_{1\le i\le m}\{p_{i[k]}\} + i\cdot\min^{(i)}_{1\le l\le m}\{p_{l[k]}\}}{m} \ge \frac{\sum_{i=1}^{m} p_{i[k]}}{m}$$
$$\ge \max_{1\le i\le m}\{p_{i[k]}\} - \min_{1\le i\le m}\{p_{i[k]}\} \ge \max_{1\le i\le m}\{p_{i[k]}\} - \min^{(i)}_{1\le i\le m}\{p_{i[k]}\}. \tag{4.32}$$

Or, equivalently,

$$\min^{(i)}_{1\le i\le m}\{p_{i[k]}\} \ge \frac{i}{m+i}\cdot p_{[k]}. \tag{4.33}$$

Therefore, by (4.30)

$$\begin{aligned} C_{[j]}(OPT) &\ge \frac{\sum_{k=1}^{j}\left(p_{[k]}+\sum_{i=1}^{m-1}\frac{i}{m+i}\cdot p_{[k]}\right)}{m} = \frac{\sum_{k=1}^{j}\left(1+\sum_{i=1}^{m-1}(1-\frac{m}{m+i})\right)\cdot p_{[k]}}{m} \\ &= \left(1-\sum_{i=1}^{m-1}\frac{1}{m+i}\right)\cdot\sum_{k=1}^{j}p_{[k]} = \left(1-\left(\sum_{i=1}^{2m-1}\frac{1}{i}-\sum_{i=1}^{m}\frac{1}{i}\right)\right)\cdot\sum_{k=1}^{j}p_{[k]} \\ &= (1+\mathcal{H}(m)-\mathcal{H}(2m-1))\cdot\sum_{k=1}^{j}p_{[k]}. \end{aligned} \tag{4.34}$$

Thus,

$$\begin{aligned} \sum w_jC_j(OPT) &= \sum_{j=1}^{n} w_{[j]}C_{[j]}(OPT) \ge (1+\mathcal{H}(m)-\mathcal{H}(2m-1))\sum_{j=1}^{n}w_{[j]}\sum_{k=1}^{j}p_{[k]} \\ &\ge (1+\mathcal{H}(m)-\mathcal{H}(2m-1))\sum_{j=1}^{n}w_j\sum_{k=1}^{j}p_k. \end{aligned} \tag{4.35}$$

By (4.29) and (4.35), the result follows. □

Note that the bound increases with m. To see this, consider the bound for $m+1$ machines and m machines, respectively. For convenience, let

$$\Gamma = (1+\mathcal{H}(m+1)-\mathcal{H}(2m+1))(1+\mathcal{H}(m)-\mathcal{H}(2m-1)).$$

The gap between the two bounds for the respective number of machines is

$$\begin{aligned} \Delta &= \frac{1}{1+\mathcal{H}(m+1)-\mathcal{H}(2m+1)} - \frac{1}{1+\mathcal{H}(m)-\mathcal{H}(2m-1)} \\ &= \left(\frac{1}{2m}+\frac{1}{2m+1}-\frac{1}{m+1}\right)/\Gamma = \frac{3m+1}{2m(m+1)(2m+1)\cdot\Gamma} > 0. \end{aligned}$$

Clearly, when $m=1$, the bound is 1; when $m=2$, the bound becomes $3/2$. Finally,

$$\lim_{m\to\infty}\frac{1}{1+\mathcal{H}(m)-\mathcal{H}(2m-1)} = \frac{1}{1-\ln 2} \approx 3.259.$$

Theorem 4.7 *For PD* $||\sum w_j C_j$, *if*

$$\sum_{i=1}^{m} p_{ij}/m \geq \delta_j$$

for each order $j = 1, 2, \ldots, n$, *then*

$$\frac{\sum w_j C_j(WECT)}{\sum w_j C_j(OPT)} \leq \frac{1}{1 + \mathcal{H}(m) - \mathcal{H}(2m-1)}.$$

Proof: The result immediately follows from Lemma 4.3 and (4.35). □

4.3.2 Additional Constraints $max_{1 \leq i \leq m}\{p_{ij}\} \leq 3min_{1 \leq i \leq m}\{p_{ij}\}$

Now change the characteristics of the processing times such that

$$\max_{1 \leq i \leq m}\{p_{ij}\} \leq 3 \min_{1 \leq i \leq m}\{p_{ij}\}, j = 1, 2, \ldots, n.$$

Actually, the constraint is equivalent to

$$\frac{\max_{1 \leq i \leq m}\{p_{ij}\} + \min_{1 \leq i \leq m}\{p_{ij}\}}{2} \leq \max_{1 \leq i \leq m}\{p_{ij}\} - \min_{1 \leq i \leq m}\{p_{ij}\}, j = 1, 2, \ldots, n.$$

With this additional constraint, the following results are obtained:

Theorem 4.8 *For PD* $||\sum w_j C_j$, *if*

$$\max_{1 \leq i \leq m}\{p_{ij}\} \leq 3 \min_{1 \leq i \leq m}\{p_{ij}\}$$

for each order $j = 1, 2, \ldots, n$, *then*

$$\frac{\sum w_j C_j(WSTP)}{\sum w_j C_j(OPT)} \leq 3 - \frac{6}{m+2}.$$

Proof: Without loss of generality, it is assumed that

$$\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \cdots \leq \frac{p_n}{w_n},$$

where $p_j = \sum_{i=1}^m p_{ij}$, $j = 1, 2, \ldots, n$. First of all, it is clear that

$$\sum w_j C_j(OPT) \geq \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} p_{[k]}/m \geq \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k/m. \tag{4.36}$$

Furthermore, it is easy to check that

$$\sum w_j C_j(WSTP) \leq \sum_{j=1}^{n} w_j \cdot \sum_{k=1}^{j} \max_i \{p_{ik}\}. \tag{4.37}$$

From the assumption that $\min_{1 \leq i \leq m}\{p_{ij}\} \geq \frac{1}{3} \max_{1 \leq i \leq m}\{p_{ij}\}$,

$$\max_{1 \leq i \leq m} \{p_{ij}\} + \frac{(m-1)\max_{1 \leq i \leq m}\{p_{ij}\}}{3} \leq \max_{1 \leq i \leq m} \{p_{ij}\} + (m-1) \min_{1 \leq i \leq m} \{p_{ij}\} \leq \sum_{i=1}^{m} p_{ij} = p_j.$$

Or, equivalently,

$$\max_{1 \leq i \leq m} \{p_{ij}\} \leq \frac{3p_j}{m+2}.$$

Thus, from (4.37)

$$\sum w_j C_j(WSTP) \leq \frac{3}{m+2} \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.38}$$

The result follows from (4.36) and (4.38). □

It is easy to see that the above bound increases monotonically in m. It is 1 when $m = 1$, and 3/2 when $m = 2$. When m becomes infinitely large, the bound is 3.

Theorem 4.9 *For $PD \,||\, \sum w_j C_j$, if*

$$\max_{1 \leq i \leq m} \{p_{ij}\} \leq 3 \min_{1 \leq i \leq m} \{p_{ij}\}$$

for each order $j = 1, 2, \ldots, n$, then

$$\frac{\sum w_j C_j(WSMP)}{\sum w_j C_j(OPT)} \leq 3 - \frac{6}{m+2}.$$

Proof: Again, it is assumed without loss of generality that

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \dots \le \frac{p_n}{w_n}, \tag{4.39}$$

where $p_j = \max_{1\le i\le m}\{p_{ij}\}$, $j = 1, 2, \dots, n$.

It is clear that

$$\sum w_j C_j(WSMP) \le \sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.40}$$

Now consider the order scheduled in the j^{th} position of an optimal solution. Its completion time is

$$C_{[j]}(OPT) = \max_{1\le i\le m}\left\{\sum_{k=1}^{j} p_{i[k]}\right\} \ge \frac{\sum_{k=1}^{j}\sum_{i=1}^{m} p_{i[k]}}{m}. \tag{4.41}$$

For each $i = 1, 2, \dots, m$

$$\min{}^{(i)}_{1\le l\le m}\{p_{l[k]}\} \ge \min_{1\le l\le m}\{p_{l[k]}\} \ge \frac{p_{[k]}}{3}. \tag{4.42}$$

The last "$\ge$" in (4.42) is due to the assumption. Therefore, from (4.41)

$$\begin{aligned} C_{[j]}(OPT) &\ge \frac{\sum_{k=1}^{j}\sum_{i=1}^{m} p_{i[k]}}{m} = \frac{\sum_{k=1}^{j}\left(\max_{1\le i\le m}\{p_{i[k]}\} + \sum_{i=1}^{m-1}\min^{(i)}_{1\le l\le m}\{p_{l[k]}\}\right)}{m} \\ &\ge \frac{\sum_{k=1}^{j}\left(p_{[k]} + \sum_{i=1}^{m-1}\frac{p_{[k]}}{3}\right)}{m} = \frac{m+2}{3m}\cdot\sum_{k=1}^{j} p_{[k]}. \end{aligned} \tag{4.43}$$

It follows that

$$\sum w_j C_j(OPT) \ge \frac{m+2}{3m}\sum_{j=1}^{n} w_{[j]}\sum_{k=1}^{j} p_{[k]} \ge \frac{m+2}{3m}\sum_{j=1}^{n} w_j \sum_{k=1}^{j} p_k. \tag{4.44}$$

The result follows from (4.40) and (4.44). □

Theorem 4.10 *For $PD\,||\sum w_j C_j$, if*

$$\max_{1\le i\le m}\{p_{ij}\} \le 3\min_{1\le i\le m}\{p_{ij}\}$$

for each order $j = 1, 2, \ldots, n$, then

$$\frac{\sum w_j C_j(WECT)}{\sum w_j C_j(OPT)} \leq 3 - \frac{6}{m+2}.$$

Proof: The result immediately follows from Lemma 4.3 and (4.44). □

4.4 LP-Based Approximation Algorithms

In this section, orders are allowed to have different release dates. Preemptions are not allowed. However, *unforced idleness* of the machines is allowed, i.e., a machine is allowed to keep idle while an operation is waiting for processing. This section presents two approximation algorithms based on different LP relaxations. For convenience of description, the following notation is defined:

$\widetilde{C}_1, \widetilde{C}_2, \ldots, \widetilde{C}_n$: The completion times in the schedule produced by an LP-based algorithm.

$C_1^*, C_2^*, \ldots, C_n^*$: The completion times in an optimal schedule.

4.4.1 Algorithm Based on a Completion Time Formulation

Hall et al. [28] presented a 3-approximation LP-based algorithm for $1 \mid r_j \mid \sum w_j C_j$. The algorithm was also considered by Chekuri and Khanna [8]. In this section, this algorithm is extended to solve $PD \mid r_j \mid \sum w_j C_j$. Thus, $PD \mid\mid \sum w_j C_j$ can also be solved as a special case.

Let $\mathcal{O} = \{1, 2, \ldots, n\}$ denote the set of all orders. For any subset $\mathcal{S} \subseteq \mathcal{O}$, let

$$p_i(\mathcal{S}) = \sum_{j \in \mathcal{S}} p_{ij}, \quad p_i^2(\mathcal{S}) = \sum_{j \in \mathcal{S}} p_{ij}^2, \qquad i = 1, 2, \ldots, m.$$

The $PD \mid r_j \mid \sum w_j C_j$ problem can be relaxed by the following linear program:

$$LP_1 = \text{ minimize } \sum_{j=1}^{n} w_j C_j,$$

subject to

$$\begin{aligned} C_j &\geq r_j + p_{ij}, \quad i = 1, \ldots, m, \ j = 1, \ldots, n; && (4.45) \\ \sum_{j \in \mathcal{S}} p_{ij} C_j &\geq \frac{p_i^2(\mathcal{S}) + (p_i(\mathcal{S}))^2}{2}, \quad i = 1, \ldots, m, \quad \text{for each } \mathcal{S} \subseteq \mathcal{O}. && (4.46) \end{aligned}$$

Constraints (4.45) are trivial. However, constraints (4.46) need some justification. Assume that $\mathcal{S} = \{1, 2, \ldots, |\mathcal{S}|\}$. It follows that for $j \in \mathcal{S}$,

$$C_j \geq C_{ij} \geq \sum_{k \leq j} p_{ik}, \quad i = 1, \ldots, m.$$

The inequality is due to the fact that there may be some idle time in the schedule because of the release dates. Thus,

$$p_{ij} C_j \geq p_{ij} C_{ij} \geq p_{ij} \sum_{k \leq j} p_{ik}.$$

Summing $p_{ij} C_j$ over all $j \in \mathcal{S}$ and simple algebra results in (4.46).

It is clear that (4.46) generates an exponential number of constraints. For the one-machine case, Queyranne [59] has shown that such constraints can be separated polynomially so that the above linear program can be solved in polynomial time by the ellipsoid method. This is the key observation that the above linear program can be used as a relaxation for approximation algorithms.

An important lemma regarding the linear programming formulation can be stated as follows:

Lemma 4.11 *Let $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_n$ be the solution to (LP_1), assume without loss of generality that $\overline{C}_1 \leq \overline{C}_2 \leq \ldots \leq \overline{C}_n$. Then, for each order $j = 1, 2, \ldots, n$,*

$$\overline{C}_j \geq \frac{1}{2} \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\}. \qquad (4.47)$$

Proof: Let $\mathcal{S} = \{1, 2, \ldots, j\}$. Constraints (4.46) imply that

$$\max_i \left\{ \sum_{k=1}^{j} p_{ik} \cdot \overline{C}_k \right\} \geq \max_i \left\{ \frac{p_i^2(\mathcal{S}) + (p_i(\mathcal{S}))^2}{2} \right\} \geq \max_i \left\{ \frac{(p_i(\mathcal{S}))^2}{2} \right\}. \qquad (4.48)$$

Since $\overline{C}_k \leq \overline{C}_j$, for each $k = 1, 2, \ldots, j$

$$\overline{C}_j \cdot \max_i \{p_i(S)\} = \max_i \left\{ \sum_{k=1}^{j} p_{ik} \overline{C}_j \right\} \geq \max_i \left\{ \sum_{k=1}^{j} p_{ik} \overline{C}_k \right\} \geq \max_i \left\{ \frac{(p_i(\mathcal{S}))^2}{2} \right\},$$

due to (4.48). Equivalently,

$$\overline{C}_j \geq \max_i \left\{ \frac{p_i(S)}{2} \right\} = \frac{1}{2} \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\}.$$

The result follows. □

Now consider the following algorithm:

An LP-based Algorithm Using Completion Times (H_{LP1})

Step 1: Solve LP_1 and let the optimal solution be $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_n$.

Step 2: Schedule the orders in nondecreasing order of $\overline{C}_j$. Ties are broken arbitrarily. Insert idle time when r_j is greater than the completion time of the $(j-1)^{th}$ order.

Assume without loss of generality that $\overline{C}_1 \leq \overline{C}_2 \leq \ldots \leq \overline{C}_n$. It is clear that

$$\sum_{j=1}^{n} w_j C_j^* \geq \sum_{j=1}^{n} w_j \overline{C}_j. \qquad (4.49)$$

In what follows, the performance guarantee of H_{LP1} will be analyzed.

Theorem 4.12 *H_{LP1} is a 2-approximation algorithm for $PD \mid\mid \sum w_j C_j$.*

Proof: Since $r_j = 0$ for all $j = 1, 2, \ldots, n$, there is no idle time in the schedule generated by H_{LP1}. For $\mathcal{S} = \{1, 2, \ldots, j\}$,

$$\widetilde{C}_j = \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\} \leq 2\overline{C}_j,$$

due to (4.47). Thus,

$$\sum_j^n w_j \widetilde{C}_j \leq 2 \sum_j^n w_j \overline{C}_j. \tag{4.50}$$

The result follows from (4.49) and (4.50). □

Theorem 4.13 *H_{LP1} is a 3-approximation algorithm for $PD \mid r_j \mid \sum w_j C_j$.*

Proof: Let $\mathcal{S} = \{1, 2, \ldots, j\}$; let $r_{max}(\mathcal{S}) = \max_{j \in \mathcal{S}}\{r_j\}$. Clearly, there is no idle time between $r_{max}(\mathcal{S})$ and $\widetilde{C}_j$. It is easy to see that,

$$\widetilde{C}_j \leq r_{max}(\mathcal{S}) + \max_i\{p_i(\mathcal{S})\}.$$

By (4.45) and the assumption $\overline{C}_1 \leq \overline{C}_2 \leq \ldots \leq \overline{C}_n$, it is easy to see that $r_{max}(\mathcal{S}) \leq \overline{C}_j$. Thus,

$$\widetilde{C}_j \leq \overline{C}_j + \max_i\{p_i(\mathcal{S})\} \leq 3\overline{C}_j, \tag{4.51}$$

due to Lemma 4.11. Therefore, by (4.49) and (4.51),

$$\frac{\sum_{j=1}^n w_j \widetilde{C}_j}{\sum_{j=1}^n w_j C_j^*} \leq 3.$$

The result follows. □

4.4.2 Algorithm Based on a Time Interval Formulation

Inspired by the time interval indexed linear programming formulation for $R \mid r_j \mid \sum w_j C_j$ due to Hall et al. [28], Wang and Cheng [66] presented a $\frac{16}{3}$-approximation algorithm for $PD \mid\mid \sum w_jC_j$. In what follows, an extension of Wang and Cheng's algorithm is proposed for solving $PD \mid r_j \mid \sum w_jC_j$.

With a given $\lambda > 1$, the time horizon of potential completion times is divided into the following intervals:

$$[1,1], (1,\lambda], (\lambda,\lambda^2], \ldots, (\lambda^{L-1},\lambda^L],$$

where L is the smallest integer such that

$$\lambda^L \geq \max_{1\leq j\leq n}\{r_j\} + \max_{1\leq i\leq m}\left\{\sum_{j=1}^{n} p_{ij}\right\}.$$

For convenience, let

$$t_0 = 1, \text{ and } t_l = \lambda^{l-1}, l = 1,\ldots,L.$$

Thus, the l^{th} interval runs from t_{l-1} to t_l, $l = 1,2,\ldots,L$. Let the decision variable x_{jl} be:

$$x_{jl} = \begin{cases} 1, & \text{if order } j \text{ is scheduled to complete within the interval } (t_{l-1},t_l]; \\ 0, & \text{otherwise.} \end{cases}$$

Consider the following linear programming relaxation:

$$LP_2 = \text{ minimize } \sum_{j=1}^{n} w_j \sum_{l=1}^{L} t_{l-1}x_{jl},$$

subject to

$$\sum_{l=1}^{L} x_{jl} = 1, \quad j = 1,\ldots,n; \tag{4.52}$$

$$\sum_{k=1}^{l}\sum_{j=1}^{n} p_{ij}x_{jk} = \sum_{j=1}^{n} p_{ij}\sum_{k=1}^{l} x_{jk} \leq t_l, \quad i = 1,\ldots,m, \; l = 1,\ldots,L; \tag{4.53}$$

$$x_{jl} = 0, \quad \text{if } t_l < r_j + p_{ij}, \quad j = 1,\ldots,n, \; l = 1,\ldots,L; \tag{4.54}$$

$$x_{jl} \geq 0, \quad j = 1,\ldots,n, \; l = 1,\ldots,L. \tag{4.55}$$

Now consider the following algorithm:

An LP-based Algorithm Using Time Intervals (H_{LP2})

Step 1: Given λ, solve LP_2 and let the optimal solution be $\bar{x}_{jl}$, $j = 1, \ldots, n$, $l = 1, \ldots, L$.

Step 2: Let $\overline{C}_j = \sum_{l=1}^{L} t_{l-1}\bar{x}_{jl}, j = 1, \ldots, n$.

Step 3: Schedule the orders in increasing order of $\overline{C}_j$. Ties are broken arbitrarily. Insert idle time when r_j is greater than the completion time of the $(j-1)^{th}$ order.

Again, assume without loss of generality that $\overline{C}_1 \leq \overline{C}_2 \leq \ldots \leq \overline{C}_n$. In what follows, the performance guarantee of H_{LP2} for solving $PD \mid r_j \mid \sum w_j C_j$ will be analyzed.

Lemma 4.14 *The optimal value of LP_2 is a lower bound for the minimum cost of $PD \mid r_j \mid \sum w_j C_j$. That is,*

$$\sum_{j=1}^{n} w_j C_j^* \geq \sum_{j=1}^{n} w_j \overline{C}_j. \tag{4.56}$$

Proof: Consider an optimal schedule π^* for $PD \mid r_j \mid \sum w_j C_j$. One can construct a solution π to LP_2 by setting $x_{jl} = 1$ if order j completes within the l^{th} interval. Clearly, π is feasible to LP_2, that is, constraints (4.52), (4.53), (4.54), and (4.55) are all satisfied. Schedule π can never be better than the optimal solution to LP_2. On the other hand, the objective cost of π^* is larger than that of π, since the completion time of order j is at least t_{l-1}. It follows that the objective cost of π^* is no better than the optimal cost of LP_2. □

Theorem 4.15 (Wang and Cheng [66]) *Given $\lambda = 2$, H_{LP2} is a $\frac{16}{3}$-approximation algorithm for $PD \mid\mid \sum w_j C_j$.*

Theorem 4.16 *Given $\lambda = 2$, H_{LP2} is a $\frac{19}{3}$-approximation algorithm for $PD \mid r_j \mid \sum w_j C_j$.*

Proof: Following the same argument as the one in Wang and Cheng [66], it can be shown that

$$\max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\} \leq \frac{16}{3} \overline{C}_j. \tag{4.57}$$

$$\widetilde{C}_j \leq \max_{1 \leq k \leq j} \{r_k\} + \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\} \leq \overline{C}_j + \frac{16}{3} \overline{C}_j \leq \frac{19}{3} \overline{C}_j. \tag{4.58}$$

Therefore,

$$\sum_{j=1}^{n} w_j \widetilde{C}_j \leq \frac{19}{3} \sum_{j=1}^{n} w_j \overline{C}_j. \tag{4.59}$$

Thus, by (4.56) and (4.59), the result follows. □

For LP_2, it would be of interest to investigate if a smaller λ leads to a better performance. In the next section, an empirical analysis will be presented.

4.5 Empirical Analyses of the Algorithms

Since the priority rules have only been formulated for $PD \mid\mid \sum w_j C_j$, the experiments are focused on the problem with all release dates equal to zero.

4.5.1 Generation of Problem Instances

For each problem size with $n = 20, 50, 100, 200$ orders and $m = 2, 5, 10, 20$ machines, 30 instances are randomly generated using a factor called *order diversity.* The order diversity κ is used to characterize the number of product types each order requests. The following three cases of order diversity are considered:

$\kappa = 2$: In problem instances 1 to 6 each order requests 2 different product types. This group is denoted by G_1.

$\kappa = m$: In problem instances 7 to 24 each order requests the maximum number of different product types, namely m, i.e, the number of machines. However, these 18 instances are grouped into the following 3 subgroups:

For instances 7 to 12 in group G_2, the processing times of each order have no additional constraints.

For instances 13 to 18 in group G_3, the processing times of each order j are subject to the constraint

$$\max_{1\le i\le m}\{p_{ij}\} \le 3 \min_{1\le i\le m}\{p_{ij}\}.$$

For instances 19 to 24 in group G_4, the processing times of each order j are subject to the constraints

$$\sum_{i=1}^{m} p_{ij}/m \ge \max_i\{p_{ij}\} - \min_i\{p_{ij}\}.$$

$\kappa = r$: In problem instances 25 to 30 in group G_5 each order requests a random number (r) of different product types; r is randomly generated from the uniform distribution $[1, m]$.

When the number of product types, l_j, for each order j is determined, l_j machines are chosen randomly. For each machine i that is selected, an integer processing time p_{ij} is generated from the uniform distribution $[1, 100]$. Note that for instances 13 to 24, the processing times of each order are generated in such a way that they satisfy the additional requirements. In addition to the generation of processing times, for each order j, a weight is randomly generated from the the uniform distribution $[1, 10]$. In total, $4 \times 4 \times 30 = 480$ instances are generated.

4.5.2 Experimental Results and Analyses

The algorithms are implemented in C++ with STL. The GLPK 4.4 (Makhorin, 2004) callable library is applied to solve the linear programs in the H_{LP2} algorithm. Since no callable library could be found for the ellipsoid method, the H_{LP1} algorithm is not implemented. The running environment is based on the Windows 2000 operating system; the PC used was a notebook computer (Pentium III 900Mhz plus 384MB RAM). It should be noted that the time-interval LP-based algorithm needs a significant amount of virtual memory. For example, for a problem instance with $n = 200$ and $m = 20$, when $\lambda = 2^{1/4}$, the total memory usage for the time-interval LP-based algorithm could reach 1GB. Due to this reason, for configuration of the virtual memory, the total file paging size for the hard disk is set to 1,152 MB.

In what follows, the performance of the algorithms is analyzed in terms of two aspects: the frequencies at which they are the best, and comparison of their average costs and average running times.

Tables 4.1 to 4.5 show the frequencies of each algorithm producing the best solution. In each table, the column labels are defined as follows:

LP_2^1: The H_{LP2} algorithm with $\lambda = 2^{1/4}$.

LP_2^2: The H_{LP2} algorithm with $\lambda = \sqrt{2}$.

LP_2^3: The H_{LP2} algorithm with $\lambda = 2$.

The data in these columns are the number of instances (out of 6 for each combination of n and m) for which the corresponding algorithms produce the best solutions.

Table 4.1 shows that for the instances of group G_1 with $k = 2$, LP_2^1 and $WECT$ have the best performance. A close study of Table 4.1 reveals that for instances with small n and large m, $WECT$ outperforms LP_2^1. However, when n becomes larger, LP_2^1 becomes significantly better than all other algorithms.

Table 4.2 shows that for the instances of group G_2, for which $k = m$ but there are no additional constraints on the properties of processing times, LP_2^1 performs the best, and $WECT$ the second best. However, LP_2^1 becomes significantly better for instances with large n.

Table 4.3 shows that $WECT$ and LP_2^1 are the two best algorithms for G_3. However, when n is small, $WSTP$, $WSMP$, $WSMC$, and $WSPL$ may occasionally beat the other algorithms. For large n, the tendency is that $WECT$ beats LP_2^1. Table 4.4 shows similar findings for G_4. These results are consistent with the theoretical

Table 4.1 The Frequency that Each Algorithm Performs the Best for Instances of Group G_1

n	m	$WSTP$	$WSMP$	$WSMC$	$WSPL$	$WECT$	LP_2^1	LP_2^2	LP_2^3
20	2	0	0	0	0	2	4	0	0
	5	0	0	0	0	3	2	1	0
	10	0	0	0	0	2	1	2	1
	20	0	0	0	0	5	1	0	0
50	2	0	0	0	0	1	5	0	0
	5	0	0	0	0	3	2	1	0
	10	0	0	0	0	1	4	1	0
	20	0	0	0	0	4	0	1	1
100	2	0	0	0	0	0	6	0	0
	5	0	0	0	0	1	5	0	0
	10	0	0	0	0	1	5	0	0
	20	0	0	0	0	1	5	0	0
200	2	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	6	0	0
	10	0	0	0	0	0	6	0	0
	20	0	0	0	0	0	6	0	0

analysis that the priority rules exhibit a better performance with additional constraints on the processing times.

Table 4.5 shows that for the instances of group G_5, for which $k = r$, LP_2^1 performs the best, and $WECT$ the second best. The tendency is that LP_2^1 becomes significantly better with large n.

The comparison among LP_2^1, LP_2^2, and LP_2^3 shows that smaller λ leads to better performance of the H_{LP2} algorithm, even though there are some exceptions.

Now the performance of the algorithms is investigated in terms of comparisons of average costs and average running times. Tables 4.6 to 4.10 show the positive ratios of the algorithms that are higher than the average costs of the best algorithms. In addition to the same field labels defined for Tables 4.1 to 4.5, four more fields are introduced:

T_1: The average running time (Seconds) per instance for all five priority rules

Table 4.2 The Frequency that Each Algorithm Performs the Best for Instances of Group G_2

n	m	$WSTP$	$WSMP$	$WSMC$	$WSPL$	$WECT$	LP_2^1	LP_2^2	LP_2^3
20	2	0	0	0	0	1	5	0	0
	5	0	0	0	0	2	4	0	0
	10	0	0	0	0	4	2	0	0
	20	0	0	0	0	2	3	0	1
50	2	0	0	0	0	1	5	0	0
	5	0	0	0	0	2	3	1	0
	10	0	0	0	0	2	4	0	0
	20	0	0	0	0	1	5	0	0
100	2	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	6	0	0
	10	0	0	0	0	1	5	0	0
	20	0	0	0	0	3	3	0	0
200	2	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	6	0	0
	10	0	0	0	0	0	6	0	0
	20	0	0	0	0	0	6	0	0

together.

T_2: The average running time (Seconds) per instance for LP_2^1.

T_3: The average running time (Seconds) per instance for LP_2^2.

T_4: The average running time (Seconds) per instance for LP_2^3.

The entries in the columns of the above four fields are simply the average running time in seconds per instance, as defined above. Now focus on the columns corresponding to the algorithms. For these columns, a "–" indicates that an algorithm produces the minimum average objective cost. Note that each row has one and only one "–" for each combination of n and m. In each row, the entries other than "–" are computed as:

$$\frac{\text{The average cost of corresponding algorithm} - \text{The minimum average cost}}{\text{The minimum average cost}} \times 100.$$

Table 4.3 The Frequency that Each Algorithm Performs the Best for Instances of Group G_3

n	m	*WSTP*	*WSMP*	*WSMC*	*WSPL*	*WECT*	LP_2^1	LP_2^2	LP_2^3
20	2	0	1	1	0	0	4	0	0
	5	0	1	0	1	2	2	0	0
	10	0	0	0	0	4	2	0	0
	20	1	0	0	0	3	0	1	1
50	2	0	0	0	0	4	2	0	0
	5	0	0	0	0	1	5	0	0
	10	0	0	0	0	4	2	0	0
	20	0	0	0	0	1	5	0	0
100	2	0	0	0	0	6	0	0	0
	5	0	0	0	0	2	4	0	0
	10	0	0	0	0	3	3	0	0
	20	0	0	0	0	3	3	0	0
200	2	0	0	0	0	6	0	0	0
	5	0	0	0	0	4	2	0	0
	10	0	0	0	0	4	2	0	0
	20	0	0	0	0	5	1	0	0

In order to rank the performance of the algorithms, in what follows the notation $\prec$ is used to indicate that algorithm A is better than algorithm B if $A \prec B$. Furthermore, if algorithm A almost ties with algorithm B, it is denoted by $A \sim B$.

First of all, Tables 4.6 to 4.10 show that, for each n, the percentage of each priority rule (except for $WECT$) tends to increase when m increases. This is consistent with the previous theoretical analysis that the performance of each priority rule becomes worse when m becomes larger. In contrast, the H_{LP2} algorithm does not have such a relationship between its performance with the value of m. Secondly, these tables also show that a smaller λ leads to a better performance of H_{LP2}. This is consistent with a previous finding from Tables 4.1 to 4.5.

Now consider the performance of the algorithms for each group of problem instances. Table 4.6 shows that, for group G_1, $WECT$ can beat LP_2^1 when $n \leq 50$. However, when $n \geq 100$, LP_2^1 is overwhelmingly better than other algorithms.

Table 4.4 The Frequency that Each Algorithm Performs the Best for Instances of Group G_4

n	m	$WSTP$	$WSMP$	$WSMC$	$WSPL$	$WECT$	LP_2^1	LP_2^2	LP_2^3
	2	0	1	1	0	2	1	0	1
	5	0	0	0	0	3	2	1	0
20	10	0	0	0	0	5	1	0	0
	20	0	0	0	0	5	0	1	0
	2	0	0	0	0	4	2	0	0
	5	0	0	0	0	4	2	0	0
50	10	0	0	0	0	4	2	0	0
	20	0	0	0	0	4	2	0	0
	2	0	0	0	0	3	3	0	0
	5	0	0	0	0	1	5	0	0
100	10	0	0	0	0	5	1	0	0
	20	0	0	0	0	5	1	0	0
	2	0	0	0	0	5	1	0	0
	5	0	0	0	0	6	0	0	0
200	10	0	0	0	0	3	3	0	0
	20	0	0	0	0	5	1	0	0

Although it is hard to fix a ranking of performance when n is small, for large n it can be seen that

$$LP_2^1 \prec LP_2^2 \prec WECT \sim LP_2^3 \sim WSMC \prec WSTP \prec WSMP \prec WSPL.$$

Table 4.7 shows that, for group G_2, $WECT$ can beat LP_2^1 when $n \leq 20$. However, when $n \geq 50$, LP_2^1 is much better than all other algorithms. For large n it can be seen that

$$LP_2^1 \prec WECT \prec LP_2^2 \prec LP_2^3 \prec WSTP \prec WSMC \prec WSMP \prec WSPL.$$

Note that the difference between $WECT$ and LP_2^1 is almost less than 1%. Thus, the performance of $WECT$ is actually quite close to that of LP_2^1. However, the table shows that LP_2^1 requires hours of running time for large instances, while $WECT$

Table 4.5 The Frequency that Each Algorithm Performs the Best for Instances of Group G_5

n	m	*WSTP*	*WSMP*	*WSMC*	*WSPL*	*WECT*	LP_2^1	LP_2^2	LP_2^3
20	2	0	0	0	0	1	4	1	0
	5	0	0	0	0	1	3	2	0
	10	0	0	0	0	2	3	0	1
	20	0	0	0	0	4	2	0	0
50	2	0	0	0	0	0	6	0	0
	5	0	0	0	0	1	3	2	0
	10	0	0	0	0	0	5	1	0
	20	0	0	0	0	1	4	0	1
100	2	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	6	0	0
	10	0	0	0	0	0	6	0	0
	20	0	0	0	0	0	6	0	0
200	2	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	6	0	0
	10	0	0	0	0	0	6	0	0
	20	0	0	0	0	0	6	0	0

requires only milliseconds. As stated before, the H_{LP2} algorithm requires virtual memory up to 1GB. By contrast, $WECT$ only requires several kilobytes of memory.

Table 4.8 shows that, for group G_3, $WECT$ can beat LP_2^1 when $n \leq 20$. However, when $n \geq 50$, LP_2^1 is overwhelmingly better than other algorithms. For large n it can be seen:

$$LP_2^1 \prec WECT \prec LP_2^2 \prec WSTP \prec WSMC \prec WSMP \prec LP_2^3 \sim WSPL.$$

Note that the difference between $WECT$ and LP_2^1 is less than 0.5%. Thus, the performance of $WECT$ is almost the same as that of LP_2^1. However, to achieve such performance, LP_2^1 requires more computational resources than $WECT$. It is also interesting to see that, $WSTP$, $WSMC$, and $WSMP$ perform better than LP_2^3.

Table 4.6 Comparison of Average Costs in Percentage and Average Running Times for Group G_1

n	m	$WSTP$	$WSMP$	$WSMC$	$WSPL$	$WECT$	LP_2^1	LP_2^2	LP_2^3	T_1	T_2	T_3	T_4
20	2	5.40	6.06	3.28	5.39	0.39	-	0.30	2.62	0.00	1.34	0.23	0.04
	5	7.80	10.05	4.45	21.64	-	2.02	1.87	2.30	0.01	4.05	0.86	0.18
	10	5.53	5.96	1.20	13.96	-	0.73	0.80	0.82	0.07	7.38	1.29	0.23
	20	16.12	14.52	12.47	22.79	-	10.09	10.09	10.15	0.07	20.77	2.63	0.46
50	2	4.30	5.39	3.37	7.11	0.64	-	0.96	1.96	0.06	9.20	1.88	0.61
	5	12.17	14.69	8.54	30.68	-	4.59	4.91	7.64	0.06	28.71	5.33	1.26
	10	5.88	7.51	2.57	24.75	7.36	-	0.29	1.13	0.09	58.01	10.02	2.03
	20	10.35	12.73	7.59	21.41	-	5.17	5.04	5.71	0.07	124	20.69	3.64
100	2	3.11	4.79	3.94	8.73	1.35	-	1.04	4.81	0.11	48.77	11.20	3.78
	5	6.59	8.09	4.35	26.66	3.68	-	0.92	4.49	0.20	121	24.50	6.73
	10	8.28	11.80	4.55	30.03	4.44	-	0.68	3.11	0.25	251	45.23	10.39
	20	7.41	9.43	3.08	24.18	4.43	-	0.19	0.95	0.26	578	98.50	17.89
200	2	2.82	4.18	3.15	8.50	0.92	-	1.14	3.85	0.24	354	87.01	32.75
	5	6.31	8.12	3.86	26.62	4.89	-	1.11	4.69	0.28	687	145	42.41
	10	6.87	8.91	4.03	28.01	6.30	-	1.00	3.99	0.30	1440	251	65.00
	20	9.01	9.59	3.77	27.46	11.34	-	0.48	2.68	0.26	4368	482	94.72

Table 4.7 Comparison of Average Costs in Percentage and Average Running Times for Group G_2

n	m	*WSTP*	*WSMP*	*WSMC*	*WSPL*	*WECT*	LP_2^1	LP_2^2	LP_2^3	T_1	T_2	T_3	T_4
	2	3.83	4.90	3.56	6.61	1.42	-	0.42	2.55	0.00	1.37	0.26	0.04
	5	3.42	4.44	3.21	12.28	-	0.03	0.44	1.64	0.01	4.95	0.68	0.13
20	10	4.13	4.94	2.71	8.12	-	0.24	0.37	1.85	0.05	18.12	1.52	0.30
	20	3.51	4.03	3.21	7.91	-	0.19	0.28	1.22	0.06	43.50	5.31	0.81
	2	3.01	4.02	2.76	8.44	0.81	-	0.57	3.04	0.07	9.34	2.05	0.65
	5	6.78	7.65	4.76	11.46	0.02	-	0.56	2.55	0.07	30.14	5.24	1.18
50	10	5.37	7.16	5.30	10.25	0.56	-	0.76	2.31	0.10	104	12.60	2.44
	20	4.42	4.41	3.71	13.34	0.16	-	0.68	1.87	0.16	518	36.80	6.42
	2	2.90	4.63	3.77	7.20	1.44	-	1.16	3.46	0.07	48.63	11.66	3.80
	5	4.80	5.95	4.29	12.63	0.93	-	1.02	3.36	0.16	139	25.00	6.22
100	10	5.64	7.44	4.94	13.52	0.40	-	1.34	2.71	0.29	503	54.50	12.02
	20	4.62	5.41	3.80	13.14	0.01	-	0.84	2.81	0.28	1994	167	26.94
	2	1.87	3.44	2.74	7.66	0.93	-	0.98	3.67	0.28	313	84.99	30.63
	5	4.34	6.62	5.30	13.71	0.96	-	1.36	3.67	0.27	855	161	47.75
200	10	5.17	6.76	5.29	13.89	0.71	-	1.18	3.13	0.25	3222	358	80.26
	20	4.15	5.41	4.26	15.29	0.65	-	1.05	2.97	0.20	15865	1199	194

Table 4.8 Comparison of Average Costs in Percentage and Average Running Times for Group G_3

n	m	$WSTP$	$WSMP$	$WSMC$	$WSPL$	$WECT$	LP_2^1	LP_2^2	LP_2^3	T_1	T_2	T_3	T_4
20	2	2.01	1.92	1.48	2.64	0.44	-	0.30	1.48	0.00	1.50	0.13	0.03
	5	1.98	1.66	1.55	3.57	0.05	-	0.25	1.01	0.00	3.70	0.30	0.05
	10	2.08	2.50	1.26	3.34	-	0.26	0.53	1.23	0.01	10.39	0.69	0.14
	20	1.57	1.80	1.38	4.09	-	0.20	0.33	1.34	0.01	24.42	1.43	0.30
50	2	1.32	1.94	1.46	3.06	0.20	-	0.44	3.76	0.01	8.51	1.12	0.38
	5	2.71	3.29	2.30	3.72	0.07	-	0.37	3.61	0.01	25.81	2.67	0.69
	10	2.57	2.90	2.19	3.62	0.13	-	0.62	2.26	0.01	60.32	3.79	0.95
	20	1.86	1.86	1.54	4.32	0.07	-	0.47	3.57	0.01	183	7.23	1.67
100	2	1.10	1.87	1.44	2.45	0.31	-	0.94	3.89	0.01	46.52	7.89	2.44
	5	2.21	2.71	2.13	3.80	0.21	-	0.95	4.04	0.02	101	14.73	3.69
	10	2.59	3.05	2.32	4.30	0.10	-	0.85	4.05	0.03	196	21.27	5.66
	20	1.98	2.27	1.85	4.03	0.03	-	0.79	4.04	0.04	783	53.20	7.96
200	2	0.90	1.59	1.26	2.38	0.12	-	0.91	3.83	0.05	284	84.13	26.02
	5	1.74	2.58	2.10	3.92	0.18	-	1.02	3.85	0.06	663	133	40.65
	10	2.04	2.68	2.18	4.13	0.15	-	0.93	3.42	0.08	2363	204	53.83
	20	1.96	2.38	2.03	4.26	0.09	-	0.83	3.22	0.20	10941	674	101

Table 4.9 shows that, for group G_4, $WECT$ is the best. The algorithms are ranked as follows:

$$WECT \prec LP_2^1 \prec LP_2^2 \prec WSTP \prec WSMC \prec WSMP \prec LP_2^3 \sim WSPL.$$

Again, $WSTP$, $WSMC$, and $WSMP$ perform better than LP_2^3. LP_2^1 could not beat $WECT$ even though it requires more computational resources.

Table 4.10 shows that, for group G_5, LP_2^1 is the best. The algorithms are ranked as:

$$LP_2^1 \prec LP_2^2 \prec WECT \prec LP_2^3 \prec WSTP \prec WSMC \prec WSMP \prec WSPL.$$

Again, the LP-based algorithm requires more computational resources than $WECT$.

To observe the cost-effective performance of the H_{LP2}, Table 4.11 compares for each group of instances the percentages that the average costs of LP_2^2 and LP_2^3 are larger than that of LP_2^1. From this table, it can be seen that the gap between the average cost of LP_2^2 and that of LP_2^1 is actually very small. For most cases, it is less than 1.0%; and the largest one is 1.3%. Therefore, the performance of LP_2^2 is actually very close to that of LP_2^1. However, from Tables 4.6 to 4.10, the average running time of LP_2^1 is much more than that of LP_2^2 (for some cases, it is more than 10 times). In addition, in the experiments, it is noticed that the memory requirement of LP_2^1 is twice that of LP_2^2. Thus, in practice, if the use of H_{LP2} is considered, it is recommended to choose $\lambda = \sqrt{2}$ in order to strike a balance between performance and the use of computational resources.

From the above empirical analysis, $WECT$ and H_{LP2} ($\lambda = \sqrt{2}$) would be good choices of algorithms for solving practical problems. Especially, in an environment which requires a solution to be generated quickly with limited memory spaces, $WECT$ is the most preferable. It is simple to implement, requires a small amount of memory, runs fast, and produces good results.

Table 4.9 Comparison of Average Costs in Percentage and Average Running Times for Group G_4

n	m	*WSTP*	*WSMP*	*WSMC*	*WSPL*	*WECT*	LP_2^1	LP_2^2	LP_2^3	T_1	T_2	T_3	T_4
20	2	1.54	1.15	0.93	1.80	-	0.02	0.15	0.36	0.00	1.39	0.17	0.03
	5	1.79	1.92	1.83	1.69	0.05	-	0.23	1.80	0.00	3.62	0.39	0.07
	10	1.61	1.97	1.45	4.28	-	0.20	0.40	2.85	0.00	15.43	1.15	0.21
	20	2.02	1.48	1.42	6.64	-	0.27	0.68	1.10	0.01	27.90	2.05	0.36
50	2	1.04	1.53	1.22	1.80	-	0.07	0.66	3.58	0.01	9.24	1.61	0.50
	5	1.99	2.56	2.26	3.33	-	0.02	0.59	2.96	0.01	23.65	3.17	0.80
	10	1.98	2.29	1.75	4.37	-	0.00	0.51	3.57	0.01	72.96	7.64	1.62
	20	1.59	2.20	1.89	6.25	-	0.12	0.73	3.00	0.02	231	15.41	3.09
100	2	0.75	1.26	1.11	1.22	-	0.01	0.96	4.63	0.01	43.95	8.76	3.66
	5	2.23	2.26	1.93	3.34	0.18	-	0.96	3.76	0.02	126	19.06	5.29
	10	2.07	2.41	2.11	3.41	-	0.10	0.82	3.75	0.03	339	31.20	7.92
	20	1.15	2.28	2.05	4.27	-	0.12	0.74	3.32	0.04	1127	98.07	16.16
200	2	0.75	1.22	1.08	1.43	-	0.13	1.03	4.74	0.05	288	76.67	26.10
	5	1.25	2.15	1.91	3.20	-	0.10	1.13	4.11	0.06	647	147	47.63
	10	1.60	2.37	2.08	4.57	0.04	-	0.87	4.08	0.08	2752	315	77.53
	20	1.29	2.26	2.05	4.84	-	0.03	0.91	3.91	0.22	13423	864	118

Table 4.10 Comparison of Average Costs in Percentage and Average Running Times for Group G_5

n	m	*WSTP*	*WSMP*	*WSMC*	*WSPL*	*WECT*	LP_2^1	LP_2^2	LP_2^3	T_1	T_2	T_3	T_4
20	2	4.97	3.95	2.36	12.82	0.03	-	0.21	1.26	0.00	1.05	0.22	0.04
	5	5.92	9.42	4.52	9.33	2.49	-	0.29	1.78	0.00	3.97	0.64	0.12
	10	5.39	7.54	5.04	16.19	0.39	-	0.23	1.71	0.07	9.93	1.52	0.31
	20	5.38	9.15	5.64	12.45	-	0.41	0.86	2.00	0.08	40.58	5.47	0.84
50	2	4.03	5.59	3.71	12.09	2.32	-	0.61	4.14	0.06	8.56	1.76	0.57
	5	7.05	9.78	7.47	15.84	2.49	-	0.49	2.81	0.07	27.17	4.59	1.08
	10	7.68	11.53	8.63	13.37	2.65	-	0.68	2.89	0.06	96.88	17.93	2.91
	20	9.03	15.40	9.31	18.17	1.86	-	0.65	2.63	0.26	324	47.02	7.04
100	2	3.01	5.02	3.39	15.01	2.12	-	1.09	5.14	0.10	44.18	10.59	3.56
	5	7.40	10.04	7.38	17.12	2.18	-	1.00	4.62	0.06	125	23.05	6.61
	10	7.99	13.35	10.72	15.86	2.40	-	0.73	3.46	0.28	373	65.16	12.25
	20	6.07	17.11	13.97	16.16	1.63	-	0.65	2.93	0.19	2594	250	42.53
200	2	2.97	4.99	3.35	17.24	2.24	-	1.15	5.14	0.25	289	78.82	27.19
	5	6.29	11.51	8.92	17.27	2.87	-	1.23	4.26	0.24	681	146	42.46
	10	6.67	14.58	11.49	16.93	2.63	-	1.01	3.67	0.29	2806	336	77.67
	20	6.64	16.54	13.84	18.71	1.76	-	1.11	3.68	0.27	16300	1229	207

Table 4.11 The Percentage that the Average Costs of LP_2^2 and LP_2^3 Are Larger Than That of LP_2^1

		G_1		G_2		G_3		G_4		G_5	
n	m	LP_2^2	LP_2^3	LP_2^2	LP_2^3	LP_2^2	LP_2^3	LP_2^2	LP_2^3	LP_2^2	LP_2^3
20	2	0.30	2.62	0.37	2.57	0.35	2.24	0.29	1.75	0.28	1.69
	5	-0.15	0.27	0.26	1.24	0.24	1.13	0.24	1.37	0.25	1.44
	10	0.07	0.09	0.12	1.33	0.17	1.17	0.18	1.65	0.19	1.62
	20	0.00	0.05	0.08	0.88	0.09	0.95	0.20	0.91	0.25	1.07
50	2	0.96	1.96	0.74	2.58	0.67	2.87	0.65	3.03	0.65	3.24
	5	0.31	2.92	0.50	2.63	0.45	3.00	0.49	2.95	0.49	2.92
	10	0.29	1.14	0.69	2.14	0.65	2.17	0.60	2.66	0.62	2.66
	20	-0.12	0.51	0.60	1.72	0.54	2.43	0.57	2.55	0.58	2.55
100	2	1.04	4.81	1.11	3.99	1.06	4.00	1.03	4.19	1.05	4.37
	5	0.92	4.49	0.99	3.62	0.97	3.80	0.97	3.78	0.98	3.96
	10	0.68	3.11	1.25	2.76	1.09	3.27	0.97	3.36	0.93	3.37
	20	0.19	0.95	0.78	2.64	0.78	3.22	0.71	3.17	0.71	3.11
200	2	1.14	3.85	1.04	3.74	1.01	3.77	0.98	4.02	1.02	4.24
	5	1.11	4.69	1.30	3.92	1.19	3.91	1.14	3.95	1.17	4.01
	10	1.00	3.99	1.16	3.25	1.07	3.33	1.00	3.60	1.01	3.59
	20	0.48	2.68	1.00	2.95	0.92	3.06	0.91	3.36	0.96	3.41

4.6 Concluding Remarks

This chapter focused on several approximation algorithms for the fully dedicated order scheduling problem with the minimization of the total weighted completion time as objective. Special interest was focused on the design of approximation algorithms for this problem. The procedures include several priority rules as well as two LP-based algorithms. Although priority rules are easy to implement, the analysis showed that their performance guarantees vary according to the distribution properties of the processing times. In contrast, various linear programming relaxations uniformly provide tight lower bounds for approximation algorithms. However, different linear programming relaxations may result in approximation algorithm with very different performance guarantees. Fortunately, both LP-based algorithms presented in this chapter have a fixed ratio performance guarantee.

Both the problems $PD \mid\mid \sum w_jC_j$ and $PD \mid r_j \mid \sum w_jC_j$ are strongly NP-hard. It is not known if there exists any polynomial time approximation scheme (PTAS) for these problems. Afrati et al. [1] presented a PTAS for the problem $1 \mid r_j \mid \sum w_jC_j$. It would be interesting to examine if their results shed any light on solving the problems studied in this chapter. On the other hand, it would also be challenging to prove that the problems are APX-hard if such PTAS does not exist unless $\mathcal{P} = \mathcal{NP}$.

With the presence of release dates, it would be interesting to investigate if the online $(3+\epsilon)$-approximation algorithm for $1 \mid r_j \mid \sum w_jC_j$ due to Hall et al. [28] can be extended to solve the online version of $PD \mid r_j \mid \sum w_jC_j$.

There is another interesting issue that comes up when orders have different release dates. In this chapter, it allowed unforced idleness, i.e., the decision-maker is allowed to keep a machine idle in anticipation of an order with a high weight coming in, even though another (old) order may be waiting for its products. Clearly, this is a reasonable assumption in certain real world applications of the order scheduling model, e.g., the equipment maintenance and repair application. However, in this chapter, it was assumed that a facility is *not* allowed to preproduce for an order that has not come in yet. In the manufacturing world, if it is known what the total quantity of product requested is (in current as well as in future orders), then the machines may be kept running producing all the different product types for orders that have not come in yet (i.e., unforced idleness is not allowed). In this case, when an order is released at a certain time, there may already be a sufficient number of the various different product types available that can be assigned to the new order, allowing for an immediate shipment. In this way, the lead times of the orders may be reduced considerably. For the LP-based algorithm using completion time formulation, it is easy to modify the formulation in such a way that it meets this requirement. For example, one only needs to change constraints (4.45) as follows:

$$C_j \geq \max\{r_j, p_{ij}\}, \quad i = 1, \ldots, m, \ \ j = 1, \ldots, n,$$

but keep the other constraints unchanged. Step 2 of this algorithm has to be changed slightly, so that it is not required to insert idle time. Simple analysis shows that, under the new assumption, H_{LP1} becomes a 2-approximation algorithm for $PD \mid r_j \mid \sum w_j C_j$. However, for the LP-based algorithm using time interval formulation, it turns out that it is not so easy to incorporate the new assumption into either the LP formulation or the algorithm.

It is believed that these types of assumptions are interesting for manufacturing and other settings. Thus, it would be of interest to consider such assumptions in future research.

CHAPTER 5

THE FULLY FLEXIBLE CASE WITH IDENTICAL MACHINES – THE TOTAL WEIGHTED COMPLETION TIME

5.1 Introduction

The previous chapters focused on the fully dedicated case of order scheduling. In this chapter and the next chapter, the fully flexible case will be considered.

Clearly, it would be more difficult to handle the fully flexible case. The reason is that, two issues have to be taken care of for this case: Besides sequencing the orders, one needs also to assign the product types to the machines. Recall that for the fully dedicated case, only the issue of sequencing the orders needs to be considered.

This chapter mainly focuses on the fully flexible case with identical machines in parallel. The objective is to minimize the total weighted completion time of the orders. According to the notation defined in Chapter 1, the problem of interest is denoted by $PF \mid \Pi k \mid \sum w_j C_j$ when k is fixed and as $PF \mid \Pi \mid \sum w_j C_j$ when k is arbitrary. Even when all $w_j = 1$, the problem with an arbitrary k is ordinary NP-hard for any fixed number (≥ 2) of machines and strongly NP-hard when the number of machines is arbitrary (see Blocher and Chhajed [4]). On the other hand, when $k = 1$, the problem $PF \mid \Pi 1 \mid \sum w_j C_j$ becomes the classical problem $P \mid\mid \sum w_j C_j$ which is strongly NP-hard (see [6]), and the problem $PFm \mid \Pi 1 \mid \sum w_j C_j$ becomes $Pm \mid\mid \sum w_j C_j$ which is ordinary NP-hard for $m = 2$ (see [7]). Thus, both $PF \mid \Pi k \mid \sum w_j C_j$ and $PFm \mid \Pi k \mid \sum w_j C_j$ are also NP-hard. Because of this, the attention is focused on the design and analysis of heuristics.

When all $w_j = 1$, Blocher and Chhajed [4] presented six heuristics for this problem and conducted an experimental study of their performance; however, they did not focus on worst-case performance bounds. One of the heuristics was also studied in Yang [67, 68]; the author established a worst-case bound for $m = 2$.

The study of the weighted version of the problem is motivated by the fact that some customers may place larger orders than other customers. Therefore, certain decision policies may require that customers be treated differently according to certain priorities or weights. This chapter presents nine approximation algorithms and compares their performance through worst-case as well as empirical analyses. The comparative study yields some interesting and counter-intuitive conclusions.

The results in this chapter also appear in Leung, Li, Pinedo [45] which has been submitted for publication. The chapter is organized as follows. Section 5.2 presents some heuristics for $PF \mid \Pi \mid \sum w_jC_j$. In Section 5.3, the performance bounds of these heuristics are analyzed. Section 5.4 compares the performance of these heuristics through an empirical analysis. Finally, some concluding remarks are presented in Section 5.5.

5.2 Heuristics

Even though $PF \mid \Pi \mid \sum w_jC_j$ is NP-hard, some easy cases can be solved polynomially. For example, if each order consists of only one product type and $w_j = 1$, then the problem can be solved by the *SPT* rule. For another example, if there is only one machine, then the problem can be solved by scheduling the orders according to the increasing order of $\sum_l p_{lj}/w_j, j = 1, 2, \ldots, n$. However, for the general problem, it would be of interest to design and analyze some effective heuristics.

Similar to the heuristics for $PF \mid \Pi \mid \sum C_j$ described in [4], it is possible to design for $PF \mid \Pi \mid \sum w_jC_j$ heuristics that consist of two phases. The first phase determines the sequence of the orders, while the second phase assigns the individual jobs within each order to the specific machines. Based on this idea, two classes of heuristics are considered: *sequential two-phase heuristics* and *dynamic two-phase heuristics.*

5.2.1 Sequential Two-Phase Heuristics

The first phase of the *sequential two-phase heuristics* sequences the orders; the second phase assigns the individual jobs of each order to the specific machines. Rules for sequencing the orders include:

- The *Weighted Shortest Total Processing time first* (*WSTP*) rule sequences the orders in increasing order of $\sum_{l=1}^{k} p_{lj}/w_j$.
- The *Weighted Shortest LPT Makespan first* (*WSLM*) rule sequences the orders in increasing order of $C_{LPT}^{(j)}/w_j$, where $C_{LPT}^{(j)}$ is the makespan of the schedule obtained by scheduling the jobs of order j on all m parallel machines according to the *longest processing time first* (*LPT*) rule, assuming each machine is available from time zero on.
- The *Weighted Shortest MULTIFIT Makespan first* (*WSMM*) rule sequences the orders in increasing order of $C_{MF}^{(j)}/w_j$, where $C_{MF}^{(j)}$ is the makespan of the schedule obtained by scheduling the jobs of order j on all m parallel machines according to the *MF* assignment rule which is described below, assuming each machine is available from time zero on.

After the sequence of the orders has been determined by one of the above rules, the individual jobs for each order are assigned to the specific machines according to one of the assignment rules listed below:

- The *List Scheduling* rule (*LS*) assigns in each iteration an unassigned (arbitrary) job to a machine with the smallest workload, until all jobs are assigned.
- The *Longest Processing Time first* rule (*LPT*) assigns in each iteration an unassigned job with the longest processing time to a machine with the smallest workload, until all jobs are assigned.
- The *Bin Packing* rule (*BIN*) first determines a target completion time for an order using the *LPT* assignment rule (just as a trial assignment). This completion time is used as a target completion time (bin size). At each iteration, the *BIN* rule assigns an unassigned job with the longest processing time to a machine with the largest workload. If the workload of the machine exceeds the target completion time after the assignment, then undo this assignment

and try the assignment on the machine with the second largest workload. This try-and-check procedure is repeated until the job can be assigned to a machine without exceeding the target completion time. If assigning the job to the machine with the smallest workload still exceeds the target completion time, then assign it to this machine, and reset the target completion time as the completion time of the job on this machine. The whole procedure is repeated until all jobs are assigned to the machines.

- The *MULTIFIT* rule (*MF*) assigns the jobs of an order to the machines following an idea that is similar to (but not exactly the same as) the *MULTIFIT* algorithm for $P \mid\mid C_{max}$ (see [11]). The original *MULTIFIT* algorithm uses the *First Fit Decreasing* (*FFD*) rule for the bin packing problem. In contrast, the *Best Fit Decreasing* (*BFD*) rule is used here. Let j be the order whose jobs are to be assigned. In the *BFD* procedure, the machines are treated as bins that are partially filled, and treat the jobs of order j as items whose sizes are exactly equal to their processing times. The jobs of order j are pre-sorted in nonincreasing order of their processing times. Given the partial schedule generated for the orders scheduled before order j, and given a target completion time t (bin size), the pre-sorted jobs of order j are assigned sequentially, each going into the bin (machine) with the largest workload in which it still fits. If all the jobs can be assigned to the machines without exceeding t, then *BFD* is considered "successful".

  Given for t a lower bound $C_L(j)$ and an upper bound $C_U(j)$, trying *BFD* with different values of t in between $C_L(j)$ and $C_U(j)$ would generate schedules of different length. If the processing times are integers, a binary search procedure would make the algorithm run faster. Using a binary search procedure, one can initially try *BFD* with $t = (C_U(j) + C_L(j))/2$; whenever *BFD* succeeds, let $C_U(j) = t$; otherwise, let $C_L(j) = t$. This procedure is repeated until t cannot be updated any more, or until after a specified number of iterations, say $\mathcal{I}$. The schedule obtained by trying *BFD* with the latest $C_U(j)$ as t is chosen.

  Now it is necessary to fix an initial setting for $C_L(j)$ and $C_U(j)$. Before the jobs of order j are assigned, let the smallest workload of the m machines be C_{min}. It is easy to see that, an initial lower bound $C_L(j)$ can be set as

$$C_L(j) = C_{min} + \max\left\{\sum_{l=1}^{k} \frac{p_{lj}}{m}, p_{1j}, \ldots, p_{kj}\right\},$$

  since $C_L(j)$ is no larger than the completion time of an optimal assignment. As for the initial upper bound $C_U(j)$, it can be set as the completion time of order j obtained by a trial assignment using the *BIN* rule. In case the *BFD* would not be successful even with the initial upper bound as its target completion time, the assignment by the *BIN* rule is accepted.

The various ways of combining sequencing rules with assignment rules lead to twelve different heuristics. However, the study is focused only on those algorithms that appear the most promising:

- Four heuristics based on *WSTP*, namely, *WSTP-LS*, *WSTP-LPT*, *WSTP-BIN*, *WSTP-MF*.
- One heuristic based on *WSLM*, namely, *WSLM-LPT*.
- One heuristic based on *WSMM*, namely, *WSMM-MF*.

The unweighted version of *WSTP-LS* has been studied in [67, 68]. The *WSTP-LPT*, *WSTP-BIN* and *WSLM-LPT* rules are generalizations of their unweighted version which are described in [4]. The *WSTP-MF* and *WSMM-MF* rules are new.

Since each heuristic consists of a sequencing rule and an assignment rule, the time complexity of a heuristic would be determined by the two rules. The time complexities of the two types of rules are considered separately. The sequencing rules are considered first.

- *WSTP* needs to compute $\sum_{l=1}^{k} p_{lj}/w_j$ for all orders, which takes $O(kn)$ time. Then, applying a sort procedure on $\sum_{l=1}^{k} p_{lj}/w_j$ takes $O(n \lg n)$ time. Thus, *WSTP* runs in $O(kn + n \lg n)$ time.
- *WSLM* needs to compute the makespan of the jobs of each order according to the *LPT* rule. Since applying *LPT* on the jobs in each order takes $O(k \lg k + k \lg m)$, n orders need $O(kn \lg km)$ time. In addition, after computing the makespans of the *LPT* schedules for all orders, the orders have to be sorted in terms of these makespans. The sorting procedure takes $O(n \lg n)$ time. Thus, the total running time of *WSLM* is $O(kn \lg km + n \lg n)$.
- *WSMM* needs to compute the makespan of the jobs of each order by using the *MF* rule. As it will be shown later, this takes $O(kn \lg k + \mathcal{I}knm)$ time. After computing the makespans, sorting the orders in terms of their makespans takes $O(n \lg n)$ time. Thus, in total, *WSMM* takes $O(kn \lg k + \mathcal{I}knm + n \lg n)$ time.

Now consider the assignment rules.

- For *LS*, a min-heap data structure can be used to maintain the machines with different workloads, it costs $O(\lg m)$ time to retrieve from the min-heap the machine with the smallest workload, and costs another $O(\lg m)$ time to update the workload of this machine in the heap after a job is assigned. Since *LS* sequentially assigns the jobs of each order in arbitrary order, and there are $O(kn)$ jobs, it follows that *LS* takes $O(kn \lg m)$ time.

- For *LPT*, additional time is required to sort the jobs of each order nonincreasingly in terms of their processing times. This takes $O(nk \lg k)$ time. The subsequential assignment procedure requires the same time as *LS*. Thus, *LPT* takes $O(nk \lg k + kn \lg m) = O(kn \lg km)$ time.

- *BIN* uses *LPT* to obtain a trial assignment, which already takes $O(kn \lg km)$ time. Note that in the worst case assigning a job needs to be tried on all m machines from the largest-workload one to the smallest-workload one. This worst-case takes $O(km)$ time to assign the jobs of each order. Therefore, n orders need $O(knm)$ time. Thus, *BIN* takes $O(kn \lg km + knm) = O(kn \lg k + knm)$ time in total.

- *MF* first uses *BIN* to determine the upper bounds of target completion times before assigning the jobs of the n orders. This takes $O(kn \lg k + knm)$ time. Then, for each order, *MF* uses *BFD* to assign its jobs. Note that in the worst case the assignment of a job according to *BFD* also needs to try all m machines from the one with the largest-workload to the one with the smallest-workload. As *BIN*, *BFD* takes $O(knm)$ time, the number of runs of *BFD* for each order is $\mathcal{I}$. Thus, *MF* takes $O(kn \lg k + \mathcal{I}knm)$ time in total. Note that $\mathcal{I}$ is usually a small integer. For example, when $\mathcal{I} = 20$, the gap between the upper bound and the lower bound is $2^{20} = 1048576$, which is a very wide range for a binary search procedure already.

The running time of each heuristic is presented in Table 5.1.

5.2.2 Dynamic Two-Phase Heuristics

The second class of heuristics are referred to as *dynamic two-phase heuristics.* In these heuristics, the sequence of the orders is not fixed prior to the assignment of the various product types to the machines, i.e., the sequence is determined dynamically. The heuristics use the *LPT* rule, the *BIN* rule or the *MF* rule to assign the jobs to the machines. However, to determine the next order to be sequenced, a greedy approach is applied to make a trial assignment of the product types of all remaining orders by using one of three rules, and the next selected order j^* satisfies

$$j^* = \arg\min_{j \in \Omega} \left\{ \frac{C_j - C_{j'}}{w_j} \right\},$$

Table 5.1 The Time Complexities for the Sequential Two-Phase Heuristics

Heuristic	Time Complexity
WSTP-LS	$O(kn \lg m + n \lg n)$
WSTP-LPT	$O(kn \lg km + n \lg n)$
WSTP-BIN	$O(kn \lg k + n \lg n + knm)$
WSTP-MF	$O(kn \lg k + n \lg n + \mathcal{I}knm)$
WSLM-LPT	$O(kn \lg km + n \lg n)$
WSMM-MF	$O(kn \lg k + n \lg n + \mathcal{I}knm)$

where Ω is the set of unscheduled orders, and $C_{j'}$ is the finish time of the order that was scheduled immediately before order j^*. Ties may be broken arbitrarily. In case $C_{j^*} < C_{j'}$, one can shift forward all those jobs of j^* assigned to each machine, and put them before all jobs of j' on that machine. Now after this shift operation, if there exists another order j'' such that $C_{j^*} < C_{j''}$, the same shift operation between j^* and j'' is carried out. This procedure is repeated until such a case does not occur any more. Clearly, with such a shift operation, the finish time of an order such as j' remains unchanged, whereas the finish time of j^* decreases. This postprocessing procedure helps to reduce the objective cost of the schedule. The three heuristics are referred to as *weighted earliest completion time by LPT* (*WECT-LPT*), *weighted earliest completion time by BIN* (*WECT-BIN*) and *weighted earliest completion time by MF* (*WECT-MF*), respectively. The first two heuristics are generalizations of the unweighted versions in [4], while *WECT-MF* is new. Natural implementation of each heuristic requires n^2 runs of the respective assignment rule. Thus, the running times of these three algorithms are $O(kn^2 \lg km)$, $O(kn^2 \lg k + kn^2 m)$ and $O(kn^2 \lg k + \mathcal{I}kn^2 m)$, respectively.

5.3 Worst-Case Analyses of the Heuristics

Let $w_jC_j(H)$ denote the weighted completion time of order j under heuristic H; let $w_jC_j(OPT)$ denote the weighted completion time of order j under the optimal schedule; and let $[j]$ denote the j^{th} order completed in the schedule. For convenience, it is also assumed without loss of generality that the orders are labeled such that

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \dots \le \frac{p_n}{w_n}, \tag{5.1}$$

where $p_j = \sum_{l=1}^{k} p_{lj}$, $j = 1, 2, \dots, n$. In what follows, the worst-case performance ratios of the above heuristics will be analyzed.

Lemma 5.1 *For the problem $PF \mid \Pi \mid \sum w_jC_j$, the worst-case performance ratio of algorithms that use LS and LPT cannot be less than $2 - \frac{1}{m}$ and $\frac{4}{3} - \frac{1}{3m}$, respectively.*

Proof: Consider $n = 1$ and $w_1 = 1$. The problem $PF \mid \Pi \mid \sum w_jC_j$ becomes $P \mid\mid C_{max}$, for which the worst-case performance ratio of LS and LPT assignment rule cannot be less than $2 - \frac{1}{m}$ (see [25]) and $\frac{4}{3} - \frac{1}{3m}$ (see [26]), respectively. □

Theorem 5.2 *For the problem $PF \mid \Pi \mid \sum w_jC_j$,*

$$\frac{\sum w_jC_j(\mathit{WSTP\text{-}LS})}{\sum w_jC_j(OPT)} \le 2 - \frac{1}{m}.$$

Proof: According to the ordering assumption in (5.1), it is easy to see that $WSTP$ sequences the orders in order of $1, 2, \dots, n$ (if not, one can always relabel the orders).

For any order j, let l^* be the product type that finishes last and determines $C_j(\mathit{WSTP\text{-}LS})$. Let the start time of l^* be s_{l^*}. Clearly,

$$s_{l^*} = C_j(\mathit{WSTP\text{-}LS}) - p_{l^*j}.$$

Since the LS rule assigns a product type to a machine with the smallest workload, all other machines must be busy when a machine starts to process l^*. It follows that

$$\begin{aligned}\sum_{x=1}^{j} p_x \geq m \cdot s_{l^*} + p_{l^*j} &= m\,(C_j(\mathit{WSTP\text{-}LS}) - p_{l^*j}) + p_{l^*j} \\ &= m \cdot C_j(\mathit{WSTP\text{-}LS}) - (m-1)p_{l^*j}.\end{aligned}$$

Or equivalently,

$$C_j(\mathit{WSTP\text{-}LS}) \leq \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \cdot p_{l^*j} \leq \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \max_{1\leq l\leq k}\{p_{lj}\}.$$

Therefore,

$$\sum_{j=1}^{n} w_j C_j(\mathit{WSTP\text{-}LS}) \leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \sum_{j=1}^{n} w_j \left(\max_{1\leq l\leq k}\{p_{lj}\}\right). \qquad (5.2)$$

On the other hand, it is clear that

$$C_{[j]}(OPT) \geq \max\left\{ \sum_{x=1}^{j} \frac{p_{[x]}}{m}, \max_{1\leq l\leq k}\{p_{l[j]}\}\right\}. \qquad (5.3)$$

Thus,

$$\begin{aligned}\sum w_{[j]} C_{[j]}(OPT) &\geq \sum_{j=1}^{n} w_{[j]} \left(\max\left\{ \sum_{x=1}^{j} \frac{p_{[x]}}{m}, \max_{1\leq l\leq k}\{p_{l[j]}\}\right\}\right) \\ &\geq \max\left\{ \sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} \frac{p_{[x]}}{m}, \sum_{j=1}^{n} w_{[j]} \left(\max_{1\leq l\leq k}\{p_{l[j]}\}\right)\right\} \\ &\geq \max\left\{ \sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{m}, \sum_{j=1}^{n} w_{[j]} \left(\max_{1\leq l\leq k}\{p_{l[j]}\}\right)\right\}, \qquad (5.4)\end{aligned}$$

the last inequality in (5.4) is due to Smith's $WSPT$ rule (see [62]).

From (5.2) and (5.4),

$$
\begin{aligned}
\frac{\sum w_j C_j(\textit{WSTP-LS})}{\sum w_j C_j(OPT)} &\le \frac{\sum_{j=1}^n w_j \sum_{x=1}^j \frac{p_x}{m} + \frac{m-1}{m}\sum_{j=1}^n w_j \left(\max_{1\le l\le k}\{p_{lj}\}\right)}{\max\left\{\sum_{j=1}^n w_j \sum_{x=1}^j \frac{p_x}{m}, \sum_{j=1}^n w_{[j]}\left(\max_{1\le l\le k}\{p_{l[j]}\}\right)\right\}} \\
&\le \frac{\sum_{j=1}^n w_j \sum_{x=1}^j \frac{p_x}{m}}{\sum_{j=1}^n w_j \sum_{x=1}^j \frac{p_x}{m}} + \frac{m-1}{m}\cdot\frac{\sum_{j=1}^n w_j\left(\max_{1\le l\le k}\{p_{lj}\}\right)}{\sum_{j=1}^n w_{[j]}\left(\max_{1\le l\le k}\{p_{l[j]}\}\right)} \\
&= 1 + \frac{m-1}{m} = 2 - \frac{1}{m}. \qquad (5.5)
\end{aligned}
$$

This completes the proof. □

It should be noted that the above bound is tight for *WSTP-LS*. (If $n = 1$ and $w_j = 1$, the problem is equivalent to $P \mid\mid C_{max}$.) Thus, the worst-case example of *List Scheduling* for $P \mid\mid C_{max}$ (see [25]) is also a worst-case example of *WSTP-LS* for $PF \mid \Pi \mid \sum w_j C_j$. Interestingly, when $w_j = 1$, it has been shown in [67, 68] that the performance bound of the unweighted version of the heuristic is also $2 - 1/m$. Thus, from the analysis above it follows that the weights do not worsen the worst-case performance of *WSTP-LS*.

Since *WSTP-LPT* is a special case of *WSTP-LS*, the following result can be obtained immediately:

Theorem 5.3 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(\textit{WSTP-LPT})}{\sum w_j C_j(OPT)} \le 2 - \frac{1}{m}.$$

This bound is not expected to be tight because of the difference between the worst case behaviors of *LPT* and *LS*. Actually, it has been shown in [67, 68] that the performance bound of the unweighted version of *WSTP-LPT* is 6/5 when $w_j = 1$ and $m = 2$, and this bound is tight.

Theorem 5.4 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(\textit{WSTP-BIN})}{\sum w_j C_j(OPT)} \le 2 - \frac{1}{m}.$$

Proof: Again, according to the ordering assumption in (5.1), it is easy to see that *WSTP* schedules the orders in the sequence $1, 2, \dots, n$. For any order j, to determine its completion time $C_j(\mathit{WSTP\text{-}BIN})$, two cases are considered.

Case 1: During the assignment, the target completion time has not been updated. In this case, the completion time of j is no later than the one obtained with the trial assignment following *LPT*. Due to the property of the *LPT* assignment rule, it is easy to show that the completion time of j obtained through *LPT* is no later than

$$\sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \max_{1 \le l \le k} \{p_{lj}\}.$$

It follows that

$$C_j(\mathit{WSTP\text{-}BIN}) \le \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \max_{1 \le l \le k} \{p_{lj}\}.$$

Case 2: During the assignment, the target completion time has been updated at least once. Consider the last time when the target completion time is updated. Let l^* denote the product type that is assigned at this particular point. Furthermore, let s_{l^*} denote the start time of l^*. Note that the completion time of l^* determines $C_j(\mathit{WSTP\text{-}BIN})$. According to the *BIN* rule, since the update of target completion time always takes place on a machine with the smallest workload, it follows that all other machines must be busy when l^* starts its processing. Thus,

$$s_{l^*} = C_j(\mathit{WSTP\text{-}BIN}) - p_{l^*j}.$$

It is clear that

$$\begin{aligned}\sum_{x=1}^{j} p_x \ge m \cdot s_{l^*} + p_{l^*j} &= m\,(C_j(\mathit{WSTP\text{-}BIN}) - p_{l^*j}) + p_{l^*j} \\ &= m \cdot C_j(\mathit{WSTP\text{-}BIN}) - (m-1)p_{l^*j}.\end{aligned}$$

Or, equivalently,

$$C_j(\mathit{WSTP\text{-}BIN}) \leq \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \cdot p_{l^*j} \leq \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \max_{1\leq l\leq k}\{p_{lj}\}.$$

From the above two cases, it follows that

$$\begin{aligned}\sum_{j=1}^{n} w_j C_j(\mathit{WSTP\text{-}BIN}) &\leq \sum_{j=1}^{n} w_j \left(\sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \cdot p_{l^*j}\right) \\ &\leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \sum_{j=1}^{n} w_j \cdot \max_{1\leq l\leq k}\{p_{lj}\}.\end{aligned}$$

From the above inequality, a similar argument for (5.5) completes the proof. □

Theorem 5.5 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(\mathit{WSTP\text{-}MF})}{\sum w_j C_j(OPT)} \leq 2 - \frac{1}{m}.$$

Proof: Note that, for each order j, the initial setting of $C_U(j)$ is the completion time of order j obtained by the trial assignment using the *BIN* rule. From Theorem 5.4,

$$C_U(j) \leq \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \max_{1\leq l\leq k}\{p_{lj}\}.$$

Therefore, the binary search behavior of the algorithm determines that

$$C_j(\mathit{WSTP\text{-}MF}) \leq C_U(j) \leq \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \max_{1\leq l\leq k}\{p_{lj}\}.$$

It follows that

$$\sum_{j=1}^{n} w_j C_j(\mathit{WSTP\text{-}MF}) \leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{m} + \frac{m-1}{m} \sum_{j=1}^{n} w_j \cdot \max_{1\leq l\leq k}\{p_{lj}\}.$$

From the above inequality, a similar argument for (5.5) completes the proof. □

It turns out that the remaining five algorithms, namely, *WSLM-LPT*, *WSMM-MF*, *WECT-LPT*, *WECT-BIN* and *WECT-MF*, can perform very badly. To see this, consider the following example:

- Let $w_j = 1, \quad j = 1, 2, \ldots, n$.
- Let $x = \rho \cdot m$, where $0 < \rho < 1$; let $\epsilon > 0$.
- Each order $j = 1, 2, \ldots, x$ requests m product types, each of which requires 1 unit of processing.
- Each order $j = x+1, x+2, \ldots, x + m(m-x)$ requests only 1 product type requiring $1+\epsilon$ units of processing.

Note that each heuristic $H \in \{$ *WSLM-LPT*, *WSMM-MF*, *WECT-LPT*, *WECT-BIN*, *WECT-MF*$\}$ produces the same schedule shown in Figure 5.1 (a), with the objective value being

$$\sum w_j C_j(H) = \sum_{j=1}^{x} j + m \cdot \sum_{j=1}^{m-x} (x + j \cdot (1+\epsilon)).$$

On the other hand, an optimal schedule for this instance is shown in Figure 5.1 (b), with the objective value being

$$\sum w_j C_j(OPT) = m \cdot \sum_{j=1}^{m-x} j \cdot (1+\epsilon) + \sum_{j=1}^{x} (j + (m-x)(1+\epsilon)).$$

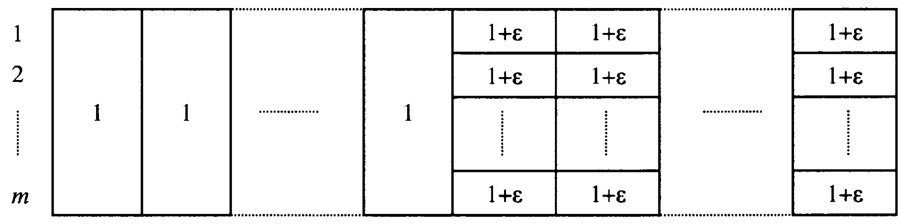
It can be determined that

$$\lim_{m \to \infty} \left(\lim_{\epsilon \to 0} \frac{\sum w_j C_j(H)}{\sum w_j C_j(OPT)} \right) = \frac{1+\rho}{1-\rho}.$$

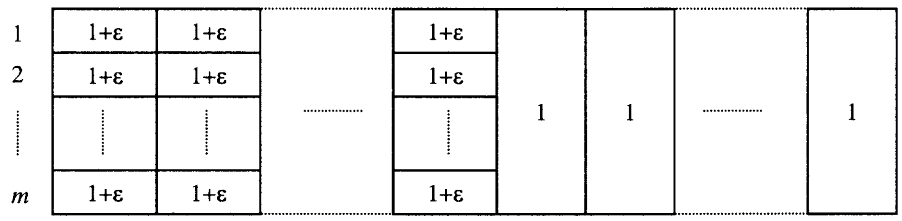
Thus, when ρ is close to 1, the above ratio can be arbitrarily large. This implies that the performance ratio of these heuristics is not bounded by any constant. Actually, in what follows, it can be shown that the upper bound is m. However, it is not clear whether this bound is tight or not. Note that the above example is not a tight example.

To make it clearer, the following concrete example is considered:

- Let $n = 96$; $m = 20$; $w_j = 1, \quad j = 1, 2, \ldots, 96$.
- Each order $j = 1, 2, \ldots, 16$ requests 20 product types, each of which requires 1 unit of processing.



(a) The schedule produced by the heuristic H



(b) An optimal schedule

Figure 5.1 A bad example for each heuristic $H \in$ {*WSLM-LPT*, *WSMM-MF*, *WECT-LPT*, *WECT-BIN*, *WECT-MF* }.

- Each order $j = 17, 18, \ldots, 96$ requests only 1 product type requiring $1 + \epsilon$ units of processing, where $\epsilon \to 0$.

For the above instance, it can be computed that

$$\frac{\sum w_j C_j(H)}{\sum w_j C_j(OPT)} = 4.04,$$

where $H \in$ {*WSLM-LPT*, *WSMM-MF*, *WECT-LPT*, *WECT-BIN*, *WECT-MF*}. In contrast, the worst-case bounds of *WSTP-LS*, *WSTP-LPT*, *WSTP-BIN* and *WSTP-MF* are never larger than 2. Thus, in a sense, the *WSTP*-based heuristics perform better than all the remaining heuristics. For this particular instance, note that the schedules produced by all *WSTP*-based heuristics are optimal.

To show the performance ratio of the *WSLM-LPT* algorithm, the following lemmas need to be proved first:

Lemma 5.6 *Suppose there are two different profiles A and B on m identical machines. If for each $i = 1, 2, \ldots m$, the i^{th} smallest workload of profile A is no larger than that of profile B, then after using LPT to assign the k product types of order j to the two different profiles,*

$$C_j(A) \leq C_j(B),$$

where $C_j(A)$ and $C_j(B)$ are the completion times of order j after its product types have been assigned to the two profiles, respectively.

Proof: The proof is relegated to Appendix B. $\square$

Lemma 5.7 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$C_{[j]}(\mathit{WSLM\text{-}LPT}) \leq \sum_{x=1}^{j} C_{LPT}^{([x])}, \quad j = 1, 2, \ldots, n.$$

Proof: The proof is by induction. Clearly, when $j = 1$,

$$C_{[1]}(\mathit{WSLM\text{-}LPT}) = C_{LPT}^{([1])}.$$

Thus, the base case holds.

As the inductive hypothesis, it is assumed that for each $j = 1, 2, \ldots, j^* - 1$

$$C_{[j]}(\mathit{WSLM\text{-}LPT}) \leq \sum_{x=1}^{j} C_{LPT}^{([x])}.$$

It certainly implies that

$$\max_{1 \leq j \leq j^*-1} \{C_{[j]}(\mathit{WSLM\text{-}LPT})\} \leq \sum_{x=1}^{j^*-1} C_{LPT}^{([x])}. \tag{5.6}$$

Based on this, it has to be shown that

$$C_{[j^*]}(\mathit{WSLM\text{-}LPT}) \leq \sum_{x=1}^{j^*} C_{LPT}^{([x])}. \tag{5.7}$$

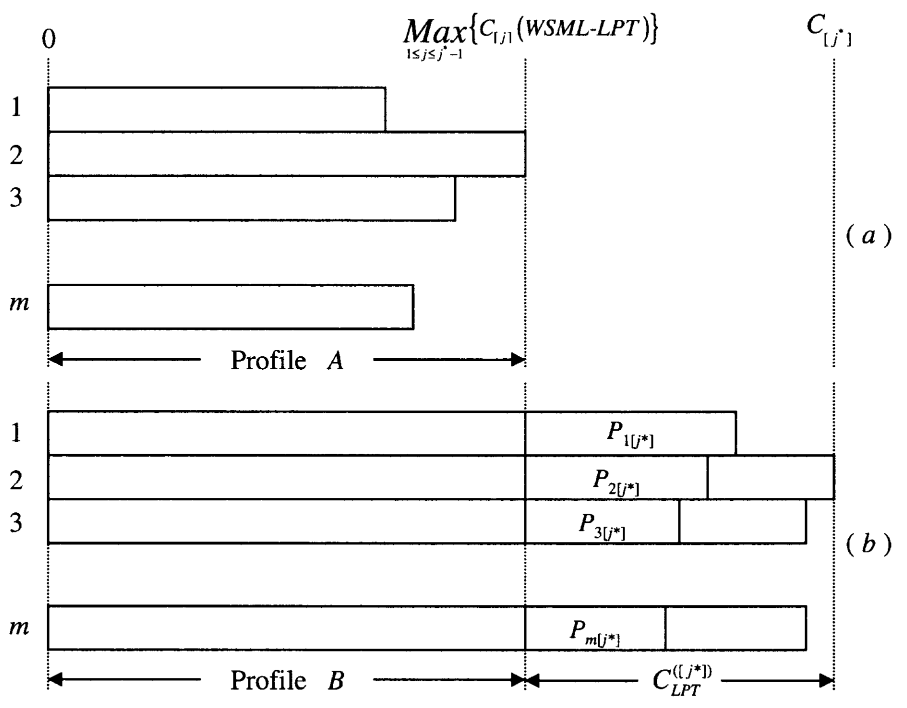


Figure 5.2 Illustrating the proof of Lemma 5.7.

Suppose that after the *WSLM-LPT* heuristic scheduled the first $j^* - 1$ orders, it produced a partial schedule of the $j^* - 1$ orders, whose profile is shown as *Profile A* in Figure 5.2 (a), in which $\max_{1\le j\le j^*-1}\{C_{[j]}(WSLM\text{-}LPT)\}$ denotes the largest completion time of the $j^* - 1$ orders. In Figure 5.2 (b), *Profile B* shows an imaginary profile in which the workload of each machine is $\max_{1\le j\le j^*-1}\{C_{[j]}(WSLM\text{-}LPT)\}$. Based on profile B, one can assign the jobs of order $[j^*]$ to the m machines using the *LPT* rule. Let $C_{[j^*]}$ be the completion time of order $[j^*]$ after the assignment. An obvious observation from Figure 5.2 (b) is that

$$C_{[j^*]} = C_{LPT}^{([j^*])} + \max_{1\le j\le j^*-1}\{C_{[j]}(WSLM\text{-}LPT)\}. \tag{5.8}$$

By Lemma 5.6, when the jobs of order j^* are assigned to profile A,

$$C_{[j^*]}(WSLM\text{-}LPT) \le C_{[j^*]}.$$

Therefore, by (5.8)

$$C_{[j^*]}(\mathit{WSLM\text{-}LPT}) \leq C_{LPT}^{([j^*])} + \max_{1\leq j\leq j^*-1}\{C_{[j]}(\mathit{WSLM\text{-}LPT})\}.$$

By (5.6)

$$C_{[j^*]}(\mathit{WSLM\text{-}LPT}) \leq C_{LPT}^{([j^*])} + \sum_{x=1}^{j^*-1} C_{LPT}^{([x])} = \sum_{x=1}^{j^*} C_{LPT}^{([x])}.$$

Thus, (5.7) holds. The result immediately follows from the inductive proof. $\square$

Lemma 5.8 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{C_{LPT}^{([j])}}{w_{[j]}} \leq \frac{p_j}{w_j}, \quad j = 1, 2, \ldots, n.$$

Proof: First of all, it should be noted for $[j]$ that

$$C_{LPT}^{([j])} \leq p_{[j]}. \tag{5.9}$$

The proof of the lemma is by contradiction. Suppose that there exists a smallest position j^* $(1 \leq j^* \leq n)$ such that

$$\frac{C_{LPT}^{([j^*])}}{w_{[j^*]}} > \frac{p_{j^*}}{w_{j^*}}. \tag{5.10}$$

When $j^* = 1$, it is clear that

$$\frac{C_{LPT}^{([1])}}{w_{[1]}} > \frac{p_1}{w_1}$$

would be impossible. Otherwise, by (5.9) it would have

$$\frac{C_{LPT}^{([1])}}{w_{[1]}} > \frac{p_1}{w_1} \geq \frac{C_{LPT}^{(1)}}{w_1}.$$

Then, according to the *WSLM* rule, order 1 should be scheduled before order [1]. This leads to a contradiction. For the case $j^* = n$,

$$\frac{C_{LPT}^{([n])}}{w_{[n]}} > \frac{p_n}{w_n}$$

would also be impossible. The reason is that p_n/w_n is the largest according to the ordering in (5.1), so there would never exist an order j such that $C_{LPT}^{([j])}/w_{[j]} > p_n/w_n$. Thus, in what follows, it is focused on $1 < j^* < n$. Due to (5.9), inequality (5.10) implies that

$$\frac{p_{[j^*]}}{w_{[j^*]}} \geq \frac{C_{LPT}^{([j^*])}}{w_{[j^*]}} > \frac{p_{j^*}}{w_{j^*}} \geq \frac{C_{LPT}^{(j^*)}}{w_{j^*}}. \tag{5.11}$$

Then, according to the ordering in (5.1),

$$[j^*] > j^*.$$

In addition, from (5.11)

$$\frac{C_{LPT}^{([j^*])}}{w_{[j^*]}} > \frac{C_{LPT}^{(j^*)}}{w_{j^*}},$$

the *WSLM* rule must have put order j^* in a position before position j^*. Thus, there must exist an order j' $(1 \leq j' < j^*)$ that has been put in a position after position j^*. Hence,

$$\frac{C_{LPT}^{([j^*])}}{w_{[j^*]}} \leq \frac{C_{LPT}^{([j'])}}{w_{[j']}} \leq \frac{p_{j'}}{w_{j'}}. \tag{5.12}$$

From (5.10) and (5.12),

$$\frac{p_{j^*}}{w_{j^*}} < \frac{p_{j'}}{w_{j'}}.$$

This leads to a contradiction, since $j' < j^*$ implies

$$\frac{p_{j^*}}{w_{j^*}} \geq \frac{p_{j'}}{w_{j'}}$$

according to the ordering in (5.1). The result follows. $\square$

Lemma 5.9 *Assume there is a sequence of n jobs scheduled on one machine, say*

$$S: \ < [1], [2], \ldots, [n] >,$$

if for each job $[j]$ $(j = 1, 2, \ldots, n)$*, its weight* $w_{[j]}$ *and processing time* $\overline{p}_{[j]}$ *satisfy*

$$w_{[j]} \in \{w_1, w_2, \ldots, w_n\}, w_{[j]} \neq w_{[j']} \text{ unless } j = j', \tag{5.13}$$

$$\overline{p}_{[j]} \leq p_{[j]}, \tag{5.14}$$

and

$$\frac{\overline{p}_{[j]}}{w_{[j]}} \leq \frac{p_j}{w_j}, \tag{5.15}$$

where p_j *and* w_j *are same as those in the ordering* (5.1)*, then*

$$\sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} \overline{p}_{[x]} \leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x.$$

Proof: To prove the above result, it is helpful to consider an artificial instance of the classical problem $1 \mid\mid \sum w_j C_j$, with jobs whose processing times and weights are given by the following pairs:

$$(p_1, w_1), (p_2, w_2), \ldots, (p_n, w_n).$$

Clearly, if one sequences the above jobs according to Smith's $WSPT$ rule, then the total weighted completion time of the schedule is

$$\sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x,$$

due to the ordering in (5.1). For convenience, this very first sequence is referred to as

$$S_1 : \ < 1, 2, \ldots, n > .$$

Now change in S_1 the processing time of the job whose label is that of the job scheduled in the first position in S, i.e., $[1]$. For convenience and consistency, the label of this job in S_1 is referred to as $[1]$. It is clear that $[1] \in \{1, 2, \ldots, n\}$. With the label $[1]$, S_1 can be rewritten as:

$$S_1 : \; < 1, 2, \ldots, [1]-1, [1], [1]+1, \ldots, n > .$$

Now, for job $[1]$ in S_1, let

$$p_{[1]} = \overline{p}_{[1]}.$$

According to (5.14), $p_{[1]}$ becomes equal to or smaller than its initial value, while $w_{[1]}$ remains unchanged. It follows that the total weighted completion time of S_1 remains unchanged or decreases after letting $p_{[1]} = \overline{p}_{[1]}$, i.e.,

$$\sum w_j C_j(S_1) \leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x. \tag{5.16}$$

Note that if $[1] = 1$, the above inequality still holds and this case is simple. Thus, it needs to focus only on the case $[1] \neq 1$. After letting $p_{[1]} = \overline{p}_{[1]}$, from (5.1) and (5.15) one can obtain a new ordering of p_j/w_j as follows:

$$\frac{p_{[1]}}{w_{[1]}} \leq \frac{p_1}{w_1} \leq \ldots \leq \frac{p_{[1]-1}}{w_{[1]-1}} \leq \frac{p_{[1]+1}}{w_{[1]+1}} \leq \ldots \leq \frac{p_n}{w_n}. \tag{5.17}$$

Swapping in S_1 the position of job $[1]$ with that of job $[1]-1$ results in a new sequence

$$S_2 : \; < 1, 2, \ldots, [1]-2, [1], [1]-1, [1]+1, \ldots, n > .$$

According to (5.17), since

$$\frac{p_{[1]}}{w_{[1]}} \leq \ldots \leq \frac{p_{[1]-1}}{w_{[1]-1}},$$

it is easy to show via an adjacent interchange argument (see Pinedo [57]) that

$$\sum w_j C_j(S_2) \leq \sum w_j C_j(S_1). \tag{5.18}$$

Repeatedly swap the position of job [1] with that of the job positioned immediately before it until the following sequence is obtained:

$$S_3 : \ < [1], 1, 2, \ldots, [1]-1, [1]+1, \ldots, n >,$$

and

$$\sum w_j C_j(S_3) \leq \sum w_j C_j(S_2). \tag{5.19}$$

The above procedure is repeated for each job $[2], [3], \ldots, [n]$ corresponding to the labels of the jobs in S, it finally produces the following sequence:

$$S_4 : \ < [1], [2], \ldots, [n-1], [n] >,$$

and

$$\sum w_j C_j(S_4) \leq \sum w_j C_j(S_3). \tag{5.20}$$

From (5.16),(5.18),(5.19) and (5.20),

$$\sum w_j C_j(S_4) \leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x. \tag{5.21}$$

Note that in S_4, for each $j = 1, 2, \ldots, n$

$$p_{[j]} = \overline{p}_{[j]}.$$

It follows that

$$\sum w_j C_j(S_4) = \sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} \overline{p}_{[x]}. \tag{5.22}$$

From (5.21) and (5.22), the result follows. □

Now it is well prepared to show the performance bound of *WSLM-LPT*.

Theorem 5.10 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(\mathit{WSLM\text{-}LPT})}{\sum w_j C_j(OPT)} \leq m.$$

Proof: : By Lemma 5.7,

$$\sum_{j=1}^{n} w_j C_j(\mathit{WSLM\text{-}LPT}) \leq \sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} C_{LPT}^{([x])}.$$

By Lemma 5.8, for each $j = 1, 2, \ldots, n$

$$w_{[j]} \in \{w_1, w_2, \ldots, w_n\}, \quad C_{LPT}^{([j])} \leq p_{[j]} \quad \text{and} \quad \frac{C_{LPT}^{([j])}}{w_{[j]}} \leq \frac{p_j}{w_j}.$$

Under the above three conditions together with the ordering in (5.1), it follows from Lemma 5.9 that

$$\sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} C_{LPT}^{([x])} \leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x.$$

Therefore,

$$\frac{\sum w_j C_j(\mathit{WSLM\text{-}LPT})}{\sum w_j C_j(OPT)} \leq \frac{\sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} C_{LPT}^{([x])}}{\sum w_j C_j(OPT)} \leq \frac{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x}{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x / m} = m.$$

The result follows. □

Lemma 5.11 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$C_{[j]}(\mathit{WSMM\text{-}MF}) \leq \sum_{x=1}^{j} C_{MF}^{([x])}, \quad j = 1, 2, \ldots, n.$$

Proof: The result follows from a similar inductive proof for Lemma 5.7. □

Lemma 5.12 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{C_{MF}^{([j])}}{w_{[j]}} \leq \frac{p_j}{w_j}, \quad j = 1, 2, \ldots, n.$$

Proof: The result follows from a similar argument for Lemma 5.8. □

Theorem 5.13 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(\mathit{WSMM\text{-}MF})}{\sum w_j C_j(OPT)} \leq m.$$

Proof: The result follows from a similar argument for Theorem 5.10. □

Theorem 5.14 *For the problem $PF \mid \Pi \mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(H)}{\sum w_j C_j(OPT)} \leq m,$$

where $H \in \{$WECT-LPT, WECT-BIN, WECT-MF$\}$.

Proof: Note that for each of these heuristics there is a postprocessing procedure. If it can be shown that the worst-case bounds of these heuristics are bounded by a ratio of m without postprocessing. Then, the result follows immediately since the postprocessing procedure helps improve the performance of each heuristic. Thus, in the following proof, the postprocessing is ignored.

For convenience, let the schedule produced by H be

$$< [1], [2], \dots, [n] > .$$

Let$C_{([0])}(H) = 0$. It can be shown by contradiction that

$$\frac{C_{[j]}(H) - C_{[j-1]}(H)}{w_{[j]}} \leq \frac{p_j}{w_j}, \quad j = 1, 2, \dots, n. \tag{5.23}$$

Suppose that there exists a smallest position j^* ($1 \leq j^* \leq n$) such that

$$\frac{C_{[j^*]}(H) - C_{[j^*-1]}(H)}{w_{[j^*]}} > \frac{p_{j^*}}{w_{j^*}}.$$

It is easy to check by contradiction that the above inequality is impossible for $j^* = 1$ or n. Thus, it needs to focus only on $1 < j^* < n$. Note that if the order scheduled in position j^* is order j^* itself, then it does satisfy (5.23) where $j = j^*$. The reason why the algorithm did not put order j^* in this position must be that it had already been scheduled earlier. However, this immediately implies that there exists at least one order j' $(1 \leq j' < j^*)$ that has not yet been scheduled by the algorithm. Now, if order j' is put in position j^*, then it satisfies

$$\frac{C_{[j^*]}(H) - C_{[j^*-1]}(H)}{w_{[j^*]}} = \frac{C_{j'}(H) - C_{[j^*-1]}(H)}{w_{j'}} \leq \frac{p_{j'}}{w_{j'}}.$$

This leads to a contradiction, since *WECT-LPT*, *WECT-BIN* and *WECT-MF* always choose as the next order the one that has the smallest

$$\frac{C_{[j^*]}(H) - C_{[j^*-1]}(H)}{w_{[j^*]}}.$$

Again, now for each $j = 1, 2, \ldots, n$

$$w_{[j]} \in \{w_1, w_2, \ldots, w_n\}, \quad C_{[j]}(H) - C_{[j-1]}(H) \leq p_{[j]} \text{ and } \frac{C_{[j]}(H) - C_{[j-1]}(H)}{w_{[j]}} \leq \frac{p_j}{w_j}.$$

Now under the above three conditions together with the ordering in (5.1), by Lemma 5.9

$$\sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} C_{[j]}(H) \leq \sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x.$$

Therefore,

$$\frac{\sum w_j C_j(H)}{\sum w_j C_j(OPT)} = \frac{\sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} C_{[x]}(H)}{\sum w_j C_j(OPT)} \leq \frac{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x}{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} p_x / m} = m.$$

The result follows. □

5.4 Empirical Analyses of the Heuristics

The problem sizes are determined by n, m and k, where $n \in \{20, 50, 100, 200\}$, $m \in \{2, 5, 10, 20\}$ and $k \in \{2, 5, 10, 20, 50, 100\}$. For each combination of n, m and k, 10 problem instances are randomly generated. These 10 problem instances have a similar structure and are treated as a group. To produce an instance for a combination of n, m and k, n orders are generated. For each order j, the number of product types k_j is generated from the uniform distribution $[1, k]$. Then, for each product type $l = 1, 2, \ldots, k_j$, an integer processing time p_{lj} is generated from the uniform distribution $[1, 100]$. In addition, a weight for order j is randomly generated from the uniform distribution $[1, 10]$. In total, $4 \times 4 \times 6 \times 10 = 960$ instances are generated.

The algorithms are implemented in C++ with STL. The running environment is based on the Windows 2000 operating system; the PC used was a notebook computer (Pentium III 900Mhz plus 384MB RAM). In what follows, the performance of the algorithms will be studied in terms of three aspects: the frequencies at which they are the best, comparison of their average costs and comparison of their average running times.

Tables 5.2 and 5.3 show the frequencies (Freq.) and the percentages (Perc.) of each heuristic performing the best for each $k \in \{2, 5, 10, 20, 50, 100\}$. Note that for some problem instances, it is possible that more than one heuristic performs the best. Thus, the sum of the frequencies may be larger than the number of problem instances for each k, i.e., 160. The two tables show that *WSTP-MF* is the best. In addition, the tendency is that a larger k leads to a higher percentage of *WSTP-MF* that are the best. The same tendency also applies to the other two heuristics based on the *MF* assignment rule, i.e., *WSMM-MF* and *WECT-MF*, even though they are inferior to *WSTP-MF*. Thus, it would be advantageous to apply *MF*-based heuristics to instances in which each order requests many different product types. When k is small, *WSMM-BIN* is comparable to *WSMM-MF*. This enforces the observation that *MF* does not have much of an advantage when each order has only a small number of jobs. Even though the analysis in the previous section showed that the worst-case performance bounds of the heuristics based on *WSTP* are much better than the other heuristics, no clear indication could be found from Tables 5.2 and 5.3. Thus, it is necessary to compare the heuristics in terms of average costs in percentage.

Table 5.4 to 5.9 show the positive ratios of the algorithms that are higher than the average costs of the best algorithms. Each table corresponds to a particular value of k. To reduce the table size, the following table fields are defined to represent the 9 heuristics:

$H1$: *WSTP-LS*	$H2$: *WSTP-LPT*	$H3$: *WSTP-BIN*
$H4$: *WSTP-MF*	$H5$: *WSLM-LPT*	$H6$: *WSMM-MF*
$H7$: *WECT-LPT*	$H8$: *WECT-BIN*	$H9$: *WECT-MF*

The entries in the columns of the above fields are simply the average objective cost per instance, over the 10 instances for each combination of k, n and m. For these columns, a "–" in a certain entry indicates that an algorithm produces the minimum average objective cost for the particular combination of k, n and m. Note that each row might have more than one "–" for each combination of n and m, the reason is that there might be more than one heuristics that produce the minimum average objective cost. In each row, the entries other than "–" are computed as:

$$\frac{\text{The average cost of corresponding algorithm} - \text{The minimum average cost}}{\text{The minimum average cost}} \times 100.$$

From these six tables, one can observe the following:

- *WSTP-MF* performs the best, *WSTP-BIN* performs second best.
- The four heuristics based on *WSTP* are better than the others. This is in agreement with the worst-case bounds obtained in the previous section.
- When k and n are fixed, the performances of the heuristics worsen as m increases. This is also consistent with the corresponding worst-case bounds obtained in the previous section.
- When n and m are fixed, the performances of the heuristics improve as k increases. In addition, as k increases, the differences in the performances of the various heuristics become smaller.

- When m and k are fixed, the differences between the *WSTP*-based heuristics become smaller as n increases. On the other hand, as k decreases, the differences between those heuristics which are not based on *WSTP* and the *WSTP*-based heuristics become larger.
- The performances of the heuristics are sensitive to k/m. When k/m is small, the gaps between the heuristics that are not based on the *WSTP* rule and the *WSTP*-based heuristics are large. When k/m is large, these gaps are small.

In order to rank the performance of the algorithms with regard to different value of k, the notation $\prec$ is used to indicate that algorithm A is better than algorithm B if $A \prec B$. Furthermore, if the performance of algorithm A is somewhat comparable to the performance of algorithm B, it is denoted by $A \sim B$.

- When $k = 2$ and 5, one can see from Table 5.4 and Table 5.5 that

  $$H3 \sim H4 \prec H2 \prec H1 \prec H6 \prec H5 \prec H8 \sim H9 \prec H7.$$

- When $k = 10$, Table 5.6 shows that

  $$H4 \prec H3 \prec H2 \prec H1 \prec H6 \prec H5 \prec H8 \sim H9 \prec H7.$$

- When $k = 20$, Table 5.7 reveals that

  $$H4 \prec H3 \prec H2 \prec H1 \prec H8 \sim H9 \prec H6 \prec H7 \prec H5.$$

- When $k = 50$, it is hard to determine a fixed ranking from Table 5.8. However, for small n and small m, it is clear that

  $$H4 \prec H3 \prec H2 \prec H9 \sim H8 \prec H7 \prec H1 \prec H6 \prec H5.$$

  On the other hand, for large n and large m, it can be seen that

  $$H4 \prec H3 \prec H2 \prec H1 \prec H9 \sim H8 \prec H7 \prec H6 \prec H5.$$

- When $k = 100$, it is also hard to determine a fixed ranking from Table 5.9. However, for small n and small m, it is clear that

  $$H4 \prec H3 \prec H2 \prec H9 \sim H8 \prec H7 \prec H6 \prec H5 \prec H1.$$

  On the other hand, for large n and large m, it can be seen that

  $$H4 \prec H3 \prec H2 \prec H1 \prec H9 \sim H8 \prec H7 \prec H6 \prec H5.$$

Table 5.2 The Frequency and Percentage that Each Heuristic Performs the Best: $k = 2, 5, 10$

	$k=2$		$k=5$		$k=10$	
Heuristic	Freq.	Perc.	Freq.	Perc.	Freq.	Perc.
WSTP-LS	0	0.00	0	0.00	0	0.00
WSTP-LPT	29	18.13	4	2.50	4	2.50
WSTP-BIN	121	75.63	85	53.13	47	29.38
WSTP-MF	116	72.50	120	75.00	125	78.13
WSLM-LPT	0	0.00	0	0.00	0	0.00
WSMM-MF	5	3.13	0	0.00	1	0.63
WECT-LPT	0	0.00	0	0.00	0	0.00
WECT-BIN	1	0.63	1	0.63	0	0.00
WECT-MF	1	0.63	0	0.00	0	0.00

Table 5.3 The Frequency and Percentage that Each Heuristic Performs the Best: $k = 20, 50, 100$

	$k=20$		$k=50$		$k=100$	
Heuristic	Freq.	Perc.	Freq.	Perc.	Freq.	Perc.
WSTP-LS	0	0.00	0	0.00	0	0.00
WSTP-LPT	2	1.25	1	0.63	1	0.63
WSTP-BIN	9	5.63	0	0.00	0	0.00
WSTP-MF	146	91.25	155	96.88	155	96.88
WSLM-LPT	0	0.00	1	0.63	1	0.63
WSMM-MF	1	0.63	8	5.00	22	13.75
WECT-LPT	0	0.00	1	0.63	1	0.63
WECT-BIN	0	0.00	0	0.00	0	0.00
WECT-MF	4	2.50	4	2.50	18	11.25

Table 5.4 Comparison of Average Costs in Percentage: $k = 2$

n	m	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*
	2	0.99	0.03	-	0.03	2.74	2.83	2.69	2.69	3.63
	5	2.17	1.00	-	-	3.31	2.27	9.59	9.09	9.09
20	10	3.01	2.21	-	-	4.19	1.22	12.61	8.48	8.48
	20	3.50	3.16	-	-	3.57	0.26	5.32	1.99	1.99
	2	0.32	-	0.001	-	3.28	3.28	4.48	4.48	4.92
	5	0.91	0.50	0.002	-	3.56	2.99	12.14	11.82	11.82
50	10	2.14	1.58	0.002	-	4.37	2.74	18.51	16.25	16.25
	20	3.80	3.44	0.002	-	5.09	1.46	14.62	10.95	10.95
	2	0.19	0.005	-	0.005	3.96	3.96	5.22	5.22	5.31
	5	0.49	0.27	0.02	-	3.81	3.52	13.80	12.22	12.22
100	10	1.19	0.92	0.006	-	3.97	2.93	23.78	21.66	21.66
	20	2.42	2.15	-	0.005	4.81	2.37	20.35	18.69	18.69
	2	0.10	0.002	-	0.002	3.78	3.78	6.17	6.17	6.00
	5	0.27	0.12	-	0.01	3.71	3.56	14.29	14.67	14.67
200	10	0.62	0.46	0.008	-	4.11	3.57	19.89	20.43	20.43
	20	1.45	1.26	-	0.01	4.53	3.10	22.82	21.65	21.65

To compare the speeds of the heuristics, Table 5.10 lists the average running times of the heuristics for $n = 200, m = 20$ and $k = 50$. The entries in the table are average running times in seconds per instance, over the 10 problem instances of $n = 200, m = 20$ and $k = 50$. From the table, it is easy to see that the *WSTP*-based heuristics run faster than other heuristics.

From all of the above observations, one would choose either *WSTP-MF* or *WSTP-BIN* to solve practical problems.

5.5 Concluding Remarks

This chapter focused on the design and analysis of nine heuristics for the scheduling of orders in a flexible environment with identical machines in parallel. The heuristics considered fall into two categories: sequential two-phase heuristics and dynamic two-phase heuristics. A worst case as well as an empirical analysis of the nine

Table 5.5 Comparison of Average Costs in Percentage: $k = 5$

n	m	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*
	2	1.52	0.24	0.03	-	1.72	1.50	1.07	0.94	1.12
	5	3.74	0.95	0.03	-	6.77	5.75	8.65	7.35	7.34
20	10	5.78	3.13	-	0.08	8.73	5.09	14.61	9.87	9.87
	20	7.92	6.11	-	-	10.07	4.16	16.15	8.04	8.04
	2	0.66	0.11	0.009	-	2.38	2.33	1.52	1.77	1.45
	5	1.58	0.31	-	0.001	8.00	7.48	7.83	7.35	6.95
50	10	3.38	2.03	0.01	-	10.55	8.20	23.01	17.74	17.45
	20	5.54	4.48	0.06	-	10.32	5.48	26.40	16.74	16.65
	2	0.30	0.06	0.01	-	2.48	2.46	1.85	1.73	1.73
	5	0.85	0.14	-	0.006	8.77	8.50	10.71	9.65	9.61
100	10	1.70	0.93	-	0.005	9.45	8.43	21.51	17.75	17.95
	20	3.36	2.65	-	0.009	9.75	6.91	24.59	20.11	20.11
	2	0.16	0.02	0.004	-	2.15	2.16	1.62	1.69	1.64
	5	0.41	0.07	0.0007	-	9.58	9.43	12.84	11.10	10.90
200	10	0.86	0.49	0.0004	-	9.92	9.33	21.41	19.39	19.00
	20	1.87	1.52	-	0.0005	10.39	8.74	27.10	25.58	25.71

heuristics was performed. The analyses reveal that the four *WSTP*-based heuristics perform better than the five other heuristics, in spite of the fact that the four *WSTP*-based heuristics are static whereas three of the other heuristics are dynamic. This may, at first sight, appear to be an anomaly, since observations in classical scheduling problems indicate that dynamic heuristics usually perform better than static heuristics. One possible reason that the static *WSTP*-based heuristics perform better than the dynamic heuristics may be the fact that the dynamic selection criteria may put some orders with many jobs ahead of a large number of orders with few jobs; the cumulative cost of these orders with few jobs becomes very large, just as in the example which was given to show that the static *WSTP*-based heuristics are better than the dynamic heuristics. The worst-case and empirical analyses also validate that the static *WSTP*-based heuristics are better than the dynamic heuristics.

Table 5.6 Comparison of Average Costs in Percentage: $k = 10$

n	m	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*
	2	1.33	0.14	0.07	-	0.78	0.64	0.42	0.25	0.27
	5	3.82	0.48	0.29	-	8.87	7.92	5.39	3.70	3.85
20	10	6.06	1.52	0.02	-	11.08	8.86	12.65	9.07	8.95
	20	7.72	4.66	0.03	-	12.57	6.69	17.25	13.45	13.51
	2	0.49	0.04	0.03	-	0.56	0.51	0.38	0.29	0.33
	5	1.66	0.31	0.13	-	8.29	7.94	5.04	3.61	4.14
50	10	2.73	0.79	0.01	-	12.85	11.64	14.01	8.09	7.99
	20	4.64	2.85	-	0.02	13.53	10.24	19.87	14.67	14.77
	2	0.27	0.03	0.01	-	0.68	0.66	0.38	0.34	0.39
	5	0.88	0.18	0.11	-	9.59	9.39	4.60	3.54	3.94
100	10	1.46	0.46	0.0002	-	13.25	12.61	14.29	11.05	10.85
	20	2.84	1.87	0.04	-	13.22	11.10	21.04	17.98	18.17
	2	0.13	0.02	0.008	-	0.65	0.65	0.35	0.30	0.30
	5	0.46	0.08	0.04	-	9.36	9.24	5.21	3.88	3.94
200	10	0.78	0.25	0.003	-	13.60	13.26	16.79	11.57	11.75
	20	1.55	1.03	-	0.0007	13.56	12.42	22.67	19.27	18.52

It was found that the performance of the heuristics depend on the k/m ratio. When k/m is small, the performance gaps between heuristics that are not based on *WSTP* and heuristics that are based on *WSTP* tend to be large; when k/m is large, these gaps tend to be small.

Another interesting observation with regard to the performance bounds of these heuristics is that when the orders have different weights their performance is not worse than when the orders have equal weights. For example, Yang and Posner [68] showed that the unweighted version of *WSTP-LS* has a tight bound of $2-1/m$; it was shown in this chapter that this tight bound also applies to *WSTP-LS* when the orders have different weights. Actually, in the analyses of the worst-case bounds, it turned out that the weights all cancelled out and the bounds were only a function of m. This may be a reason why the inclusion of weights does not worsen the performance of the heuristics.

Table 5.7 Comparison of Average Costs in Percentage: $k = 20$

n	m	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*
	2	0.66	0.04	0.04	-	0.21	0.17	0.08	0.06	0.06
	5	3.22	0.34	0.43	-	3.03	2.64	1.55	0.88	0.60
20	10	5.84	0.79	0.43	-	13.21	12.35	7.17	3.98	3.69
	20	7.67	2.27	0.13	-	15.72	11.93	16.84	9.08	9.20
	2	0.33	0.02	0.02	-	0.13	0.10	0.09	0.07	0.07
	5	1.48	0.21	0.17	-	3.13	3.06	1.37	1.06	0.87
50	10	2.54	0.56	0.20	-	13.63	12.93	5.68	4.09	4.51
	20	3.95	1.38	0.06	-	14.17	12.36	13.74	8.56	8.38
	2	0.16	0.01	0.007	-	0.16	0.14	0.09	0.07	0.08
	5	0.74	0.10	0.08	-	3.04	3.07	1.15	0.64	0.67
100	10	1.36	0.27	0.11	-	13.16	12.73	6.55	2.99	3.82
	20	2.20	0.86	0.02	-	16.45	15.28	16.95	9.96	10.19
	2	0.09	0.007	0.004	-	0.17	0.16	0.08	0.09	0.07
	5	0.37	0.05	0.04	-	3.09	3.17	1.27	0.76	0.86
200	10	0.69	0.13	0.05	-	14.46	14.20	7.72	3.41	3.60
	20	1.11	0.43	0.01	-	16.57	15.93	16.21	10.08	9.98

The analyses are helpful to rank these heuristics according to their effectiveness, taking solution quality as well as running time into account. The *WSTP*-based heuristics clearly dominate.

A number of questions still remain open. It would be of interest to obtain tighter bounds for the *WSTP*-based heuristics. The work of Yang and Posner [68] may provide some insights in how to obtain tighter bounds. However, it should be noted that their arguments become significantly more complicated when $m > 2$ and $w_j \neq 1$. Nevertheless, it still may be possible to obtain a better bound like 4/3 for *WSTP-LPT*, *WSTP-BIN* and *WSTP-MF*.

For the other heuristics that are not based on the *WSTP* rule, bounds may exist that are a tighter function (sublinear, for example) of m. From the example illustrated in Figure 5.1, it is easy to find instances with ratios of 2.7561 for $m = 10$,

Table 5.8 Comparison of Average Costs in Percentage: $k = 50$

n	m	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*
	2	0.38	0.02	0.01	-	0.06	0.04	0.04	0.03	0.03
	5	1.60	0.12	0.11	-	0.46	0.34	0.21	0.18	0.08
20	10	3.58	0.44	0.38	-	3.11	2.82	1.92	1.29	1.22
	20	6.74	0.94	0.59	-	13.60	12.83	6.44	4.44	4.20
	2	0.16	0.008	0.006	-	0.04	0.03	0.03	0.03	0.02
	5	0.73	0.06	0.06	-	0.52	0.46	0.23	0.21	0.25
50	10	1.66	0.21	0.19	-	2.93	2.86	1.03	0.61	0.62
	20	3.15	0.51	0.24	-	15.14	14.88	5.82	3.25	3.34
	2	0.08	0.003	0.003	-	0.01	0.01	0.01	0.008	0.008
	5	0.39	0.04	0.03	-	0.36	0.33	0.17	0.13	0.10
100	10	0.86	0.11	0.09	-	2.77	2.80	1.08	0.52	0.57
	20	1.59	0.27	0.12	-	14.26	14.10	6.96	2.68	3.38
	2	0.04	0.002	0.002	-	0.02	0.01	0.009	0.007	0.007
	5	0.19	0.02	0.02	-	0.46	0.45	0.18	0.11	0.12
200	10	0.43	0.06	0.04	-	2.57	2.67	0.80	0.42	0.48
	20	0.82	0.14	0.06	-	14.72	14.79	6.15	2.68	2.89

4.04 for $m = 20$, 6.6146 for $m = 50$, 9.5304 for $m = 100$, and 31.1294 for $m = 1000$, respectively. Thus, the ratio goes up slowly when m increases.

It may be possible to design LP-based approximation algorithms for this problem. However, since the *WSTP*-based heuristics already have worst-case bounds that are less than 2, the extensive amount of computing time would discourage the use of an LP-based approximation algorithm if its worst-case bound is not significantly better than 2.

It would be interesting to consider more general machine environments, for example, uniform machines and unrelated machines. It has been noticed that no one has ever considered yet such machine environments for order scheduling. In the next chapter, uniform machines will be considered.

Table 5.9 Comparison of Average Costs in Percentage: $k = 100$

n	m	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*
	2	0.20	0.007	0.005	-	0.007	-	0.007	0.006	0.0007
	5	0.90	0.04	0.05	-	0.16	0.11	0.19	0.18	0.15
20	10	2.33	0.18	0.18	-	1.03	0.90	0.57	0.40	0.22
	20	4.56	0.50	0.45	-	3.14	2.73	2.28	1.41	1.04
	2	0.08	0.003	0.002	-	0.005	0.001	0.005	0.003	0.004
	5	0.40	0.02	0.02	-	0.07	0.05	0.06	0.03	0.03
50	10	0.94	0.08	0.08	-	0.63	0.58	0.38	0.27	0.31
	20	2.06	0.25	0.20	-	4.27	4.33	1.96	1.01	1.07
	2	0.04	0.001	0.001	-	0.003	0.001	0.003	0.002	0.0009
	5	0.21	0.01	0.01	-	0.07	0.06	0.03	0.02	0.02
100	10	0.49	0.04	0.04	-	0.60	0.58	0.28	0.14	0.15
	20	1.03	0.14	0.11	-	3.72	3.84	1.57	0.68	0.96
	2	0.02	0.0006	0.0005	-	0.002	0.001	0.002	0.0010	0.0008
	5	0.11	0.006	0.006	-	0.07	0.06	0.02	0.02	0.02
200	10	0.26	0.02	0.02	-	0.70	0.70	0.23	0.12	0.13
	20	0.54	0.07	0.05	-	4.52	4.75	1.55	0.62	0.75

Table 5.10 Comparison of Average Running Times over the 10 Problem Instances of $n = 200, m = 20$ and $k = 50$

Heuristic	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*
Avg time (Sec.)	0	0	0.008	0.018	0.002	0.020	0.157	0.537	2.255

Another interesting issue concerns setup times when a machine switches over from one product type to another. This case occurs in many practical applications. However, setup times inevitably make the problem harder.

CHAPTER 6

THE FULLY FLEXIBLE CASE WITH UNIFORM MACHINES – THE TOTAL WEIGHTED COMPLETION TIME

6.1 Introduction

Chapter 5 investigated the fully flexible case with m identical machines in parallel. This chapter considers the fully flexible case with m uniform machines in parallel.

The uniform machines are assumed to have different speeds. Let v_i be the speed of machine $i = 1, 2, \ldots, m$. Thus, in one unit of time machine i can carry out v_i units of processing. Without loss of generality, it is assumed that

$$v_1 \geq v_2 \geq \ldots \geq v_m.$$

Each order $j = 1, 2, \ldots, n$ requests a certain quantity of product type $l = 1, \ldots, k$ and the units of processing required is $p_{lj} \geq 0$. For convenience, it may be assumed that

$$p_{1j} \geq p_{2j} \geq \ldots \geq p_{kj}.$$

As mentioned in Chapter 1, in the fully flexible case, the items of type l can be produced on any one of the m machines. If the items of type l is produced on machine i, then the units of processing time on machine i is $p_{lij} = p_{lj}/v_i$. The k different jobs of order j may be processed concurrently. Preemptions are allowed. In addition, it is assumed that each order $j = 1, 2, \ldots, n$ is released at time zero and has a weight w_j.

The objective of interest is minimizing the total weighted completion time of the n orders, namely $\sum w_j C_j$. According to the notation defined in Chapter 1, for the non-preemptive case, the problem with a fixed k is denoted by $QFm \mid \Pi k \mid \sum w_j C_j$

when the number of machines m is fixed, and as $QF \mid \Pi k \mid \sum w_jC_j$ when the number of machines m is arbitrary. If k is arbitrary, then the k is dropped from the notation.

It is clear that when $n = 1$, $QF \mid \Pi \mid \sum C_j$ becomes $Q \mid\mid C_{\max}$, which is known to be NP-hard in the strong sense because of a reduction from the 3-Partition problem (see Garey and Johnson [19]). It follows that $QF \mid \Pi \mid \sum w_jC_j$ is also NP-hard in the strong sense. As discussed in Chapter 5, $PF \mid \Pi k \mid \sum w_jC_j$ is strongly NP-hard. Therefore, $QF \mid \Pi k \mid \sum w_jC_j$ is also strongly NP-hard since it is a generalization of $PF \mid \Pi k \mid \sum w_jC_j$. Similarly, both $QFm \mid \Pi \mid \sum w_jC_j$ and $QFm \mid \Pi k \mid \sum w_jC_j$ are NP-hard due to the NP-hardness of $PFm \mid \Pi \mid \sum w_jC_j$ and $PFm \mid \Pi k \mid \sum w_jC_j$. Thus, it would be of interest to develop effective heuristics for the nonpreemptive case. As just mentioned, when $n = 1$, the problem $QF \mid \Pi \mid \sum C_j$ is actually $Q \mid\mid C_{\max}$. For this special case, Gonzalez, Ibarra and Sahni [24] showed that the worst-case ratio of the *LPT* rule is $2m/(m+1)$. Dobson [13] showed that *LPT* has a tighter bound that lies in between $[1.512, 1.583]$, while Friesen [18] showed that the bound lies in between $[1.52, 1.67]$.

When all machines are identical, i.e., $v_1 = v_2 = \ldots = v_m$, Blocher and Chhajed [4] presented six heuristics and conducted an experimental study of their performance; however, they did not focus on worst-case performance bounds. One of the heuristics was also analyzed by Yang [67] and Yang and Posner [68]; they obtained a worst-case bound of $7/6$ for $m = 2$. For the same problem with the additional feature that each order has a weight, four $(2 - 1/m)$-approximation algorithms and five m-approximation algorithms were presented in Chapter 5.

One can define two forms of preemptions for this problem. According to one form of preemption multiple machines can work simultaneously on the same product for a given order; a problem that allows this form of preemption is easy. The orders are first sequenced in ascending order of their total weighted processing times. Then, for each job l of order j, assign $p_{lj} * v_i / \sum_{l=1}^{m} v_l$ on machine i. Via this method an optimal schedule is obtained.

A second form of preemptions does not allow multiple machines to process simultaneously the same product for an order; however, it does allow the processing of a product for a given order to be interrupted and resumed at a later point in time, possibly on a different machine. Assuming this form of preemption, the problem is NP-hard even with two identical machines, $k = 2$, and all $w_j = 1$ (see Leung et al. [44]). In what follows, only this second form of preemption will be considered. According to the notation defined in Chapter 1, the problem with arbitrary number of machines is referred to as $QF \mid \Pi k, prmp \mid \sum w_j C_j$ when k is fixed and as $QF \mid \Pi, prmp \mid \sum w_j C_j$ when k is arbitrary. When m is fixed, QF is replaced with QFm in the notation.

This chapter presents an approximation algorithm for solving the nonpreemptive version and the preemptive version of this problem, respectively; the algorithm for the nonpreemptive version is denoted by H_{NP} and the algorithm for the preemptive version is denoted by H_P. Both heuristics are shown to have a worst-case bound of m. In addition, when $m = 3$, it is shown that H_P has a worst-case bound of 2. Finally, an empirical analysis is performed for the two heuristics.

The following notation is adopted. Let $C_j(H)$ denote the completion time of order j under heuristic H, where H is either H_{NP} or H_P. Let $C_j(OPT)$ denote the completion time of order j under the optimal schedule, and let $[j]$ denote the j^{th} order completed in the optimal schedule. It is also assumed without loss of generality that the orders are labeled such that

$$\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_n}{p_n}, \tag{6.1}$$

where $p_j = \sum_{l=1}^{k} p_{lj}$, $j = 1, 2, \ldots, n$. Furthermore, the sum of the speeds of the m uniform machines is denoted by

$$v = \sum_{i=1}^{m} v_i.$$

This chapter contains the results that appear in Leung et al. [50] which has been submitted for publication. It is organized as follows. The next section presents an approximation algorithm for $QF \mid \Pi \mid \sum w_j C_j$ as well as its worst-case analysis. Section 6.3 presents an approximation algorithm for $QF \mid \Pi, prmp \mid \sum w_j C_j$ as well as its worst-case analysis. Section 6.4 performs an empirical analysis for the two algorithms. Finally, Section 6.5 contains some concluding remarks.

6.2 Approximation Algorithm for the Nonpreemptive Case

It seems that no one has ever presented any algorithms for $QF \mid \Pi \mid \sum w_j C_j$. This section presents for this problem a heuristic that consists of two phases. The first phase determines the sequence of the orders, while the second phase assigns the individual jobs within each order to the specific machines. In the first phase, the following rule is used to sequence the orders:

> The *Weighted Shortest Total Processing time first* (*WSTP*) rule sequences the orders in increasing order of $(\sum_{l=1}^{k} p_{lj})/w_j$. Ties are broken arbitrarily.

After the sequence of orders has been determined by the rule above, the individual jobs for each order are assigned to the specific machines according to the following assignment rule:

> The *Longest Processing Time first* rule (*LPT*) selects in each iteration among the remaining jobs a job with the longest processing time and assigns it to a machine on which it has the earliest finish time.

It is clear that the *WSTP* rule takes $O(kn + n \log n)$ time. On the other hand, assignment of a job by the *LPT* rule takes $O(m)$ time. n orders have at most nk

jobs. Thus, the *LPT* rule takes $O(mnk)$ time. Therefore, the whole algorithm takes $O(mnk + n\log n)$ time.

The two-phase algorithm is denoted by H_{NP}. Now consider the worst-case performance ratio of the algorithm. The following result from Horvath, Lam and Sethi [31] is made use of:

Theorem 6.1 *For a set of independent jobs with processing requirement* $p_1 \geq p_2 \geq \ldots \geq p_n$*, the level algorithm due to Horvath, Lam and Sethi [31] constructs a minimal length preemptive schedule on* m *uniform machines with speeds* $v_1 \geq v_2 \geq \ldots \geq v_m$*. The makespan is given by*

$$\omega = \max\left\{\frac{\sum_{x=1}^{n} p_x}{\sum_{i=1}^{m} v_i}, \max_{1\leq i<m}\left\{\frac{\sum_{x=1}^{i} p_x}{\sum_{x=1}^{i} v_x}\right\}\right\}.$$

For the same set of independent jobs, it is clear that the makespan of an optimal non-preemptive schedule will never be less than ω. Thus, ω is also a lower bound of the makespan of any non-preemptive schedule.

Theorem 6.2 *For the problem* $QF \mid \Pi \mid \sum w_j C_j$*,*

$$\frac{\sum w_j C_j(H_{NP})}{\sum w_j C_j(OPT)} \leq m.$$

Proof: In the first phase of H_{NP}, it can be assumed that the orders are sequenced as

$$< 1, 2, \ldots, n > .$$

If not, one can always relabel the orders. For any order j, let l^* be the product type that finishes on machine i^* and then determines $C_j(H_{NP})$. Let P_i denote the total amount of processing already assigned to machine i. So P_i/v_i is the finish time of machine i ($1 \leq i \leq m$) before job l^* is assigned. The profile is shown in Figure 6.1.

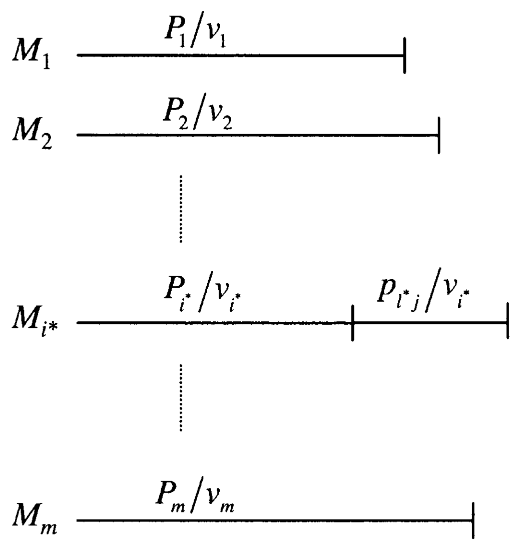


Figure 6.1 The profile after l^* is assigned

Since l^* determines $C_j(H_{NP})$, it follows that

$$C_j(H_{NP}) = \frac{P_{i^*} + p_{l^*j}}{v_{i^*}}. \tag{6.2}$$

In addition, according to the *LPT* rule, l^* is assigned to the machine on which it has the earliest completion time (ties may be broken arbitrarily). Thus, if assign l^* to any machine $i \neq i^*$, then the finish time of l^* would be at least $(P_{i^*} + p_{l^*j})/v_{i^*}$. Therefore,

$$C_j(H_{NP}) \leq \frac{P_i + p_{l^*j}}{v_i}, \text{ for each } i \neq i^*. \tag{6.3}$$

From (6.2) and (6.3),

$$\begin{aligned}\left(v_{i^*} + \sum_{i \neq i^*} v_i\right) C_j(H_{NP}) &\leq \sum_{i=1}^{m} P_i + m \cdot p_{l^*j} = \sum_{h=1}^{j} p_h - \sum_{l=l^*}^{k} p_{lj} + m \cdot p_{l^*j} \\ &\leq \sum_{h=1}^{j} p_h + (m-1)p_{l^*j} \leq \sum_{h=1}^{j} p_h + (m-1)p_{1j}. \end{aligned} \tag{6.4}$$

Or, equivalently,

$$C_j(H_{NP}) \leq \frac{\sum_{h=1}^{j} p_h}{v} + \frac{(m-1) \cdot p_{1j}}{v} \leq \frac{\sum_{h=1}^{j} p_h}{v} + \frac{(m-1) \cdot p_{1j}}{v_1}. \tag{6.5}$$

On the other hand, let $\overline{p}_1 \geq \overline{p}_2 \geq \ldots \geq \overline{p}_{m-1}$ denote the processing requirements of the longest $m-1$ jobs among the first j orders in the optimal schedule. According to Lemma 6.1, it is clear that

$$\begin{aligned} C_{[j]}(OPT) &\geq \max\left\{\frac{\sum_{h=1}^{j} p_{[h]}}{\sum_{i=1}^{m} v_i}, \max_{1\leq i<m}\left\{\frac{\sum_{l=1}^{i} \overline{p}_l}{\sum_{l=1}^{i} v_l}\right\}\right\} \\ &\geq \max\left\{\sum_{h=1}^{j} \frac{p_{[h]}}{v}, \max_{1\leq i<m}\left\{\frac{\sum_{l=1}^{i} p_{l[j]}}{\sum_{l=1}^{i} v_l}\right\}\right\} \\ &\geq \max\left\{\sum_{h=1}^{j} \frac{p_{[h]}}{v}, \frac{p_{1[j]}}{v_1}\right\}, \qquad (6.6) \end{aligned}$$

since $\overline{p}_l \geq p_{l[j]}$ for each $l = 1, 2, \ldots, m$.

Therefore, from (6.5) and (6.6)

$$\begin{aligned} \frac{\sum w_j C_j(H_{NP})}{\sum w_j C_j(OPT)} &\leq \frac{\sum_{j=1}^{n} w_j \left(\frac{\sum_{h=1}^{j} p_h}{v} + \frac{(m-1)\cdot p_{1j}}{v_1}\right)}{\sum_{j=1}^{n} w_{[j]} \cdot \max\left\{\sum_{h=1}^{j} \frac{p_{[h]}}{v}, \frac{p_{1[j]}}{v_1}\right\}} \\ &\leq \frac{\sum_{j=1}^{n} w_j \sum_{h=1}^{j} \frac{p_h}{v}}{\sum_{j=1}^{n} w_{[j]} \sum_{h=1}^{j} \frac{p_{[h]}}{v}} + \frac{(m-1)\sum_{j=1}^{n}(w_j \cdot p_{1j})/v_1}{\sum_{j=1}^{n}(w_{[j]} \cdot p_{1[j]})/v_1} \\ &= 1 + (m-1) = m. \qquad (6.7) \end{aligned}$$

The last inequality in (6.7) is due to the *WSPT* rule (see [62]) and the ordering in (6.1). This completes the proof. □

To see that H_{NP} could perform badly, the following example is considered:

- Let $n = 4$ and $w_j = 1, \;\; j = 1, 2, 3, 4$.
- let $k = m^2 + 1$.
- There are $m^2 + 1$ machines, with speeds: $v_1 = m, v_2 = v_3 = \ldots = v_{m^2+1} = 1$.
- The processing times of the 4 orders are:

  $p_{11} = m^2, p_{21} = p_{31} = \ldots = p_{k1} = 0$;

  $p_{1j} = 2m, p_{2j} = p_{3j} = \ldots = p_{kj} = 1$, for each $j = 2, 3, 4$.

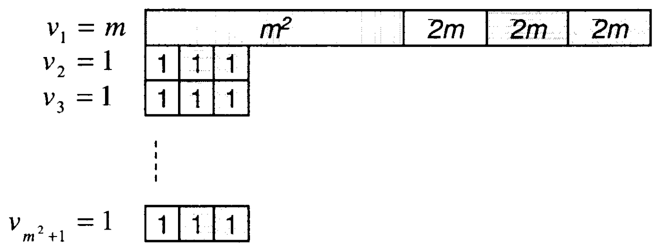


Figure 6.2 The schedule produced by H_{NP}.

When m is sufficiently large, it is clear that the schedule produced by H_{NP} is the one shown in Figure 6.2. Thus,

$$\sum_{j=1}^{4} C_j(H_{NP}) = m + (m+2) + (m+4) + (m+6) = 4m + 12.$$

On the other hand, in the schedule shown in Figure 6.2, if put order 2, 3, and 4 before order 1, then it results in an optimal schedule whose objective cost is:

$$\sum_{j=1}^{4} C_j(OPT) = 2 + 4 + 6 + (m+6) = m + 18.$$

Therefore, when $m \to \infty$,

$$\lim_{m\to\infty}\left(\frac{\sum_{j=1}^{4} C_j(H_{NP})}{\sum_{j=1}^{4} C_j(OPT)}\right) = \lim_{m\to\infty}\left(\frac{4m+12}{m+18}\right) = 4.$$

It is easy to generalize this example and generate a series of instances for which the worst case ratio increases in $O(\sqrt{m})$.

6.3 Approximation Algorithm for the Preemptive Case

Again, heuristics for $QF \mid \Pi, prmp \mid \sum w_j C_j$ may consist of two phases. In the first phase, one can use the *WSTP* rule to sequence the orders according to their total weighted processing times. In the second phase, the jobs of an order are assigned using the following algorithm which is based on the *level algorithm* (see Muntz and Coffman [52, 53]) generalized by Horvath, Lam and Sethi [31] for preemptive scheduling on uniform machines.

For any order j, let $L_t(l, j)$ denote the *level* of job l of order j at time point t. Since the jobs are independent, the initial level for a job l of order j is set to be $L_t(l, j) = p_{lj}$ before the jobs of order j are assigned.

The following procedure is used to construct a *shared* schedule in which the machines are shared equally among jobs. The shared schedule is used to construct a preemptive schedule in a later procedure.

PROCEDURE *Shared-Schedule*

INPUT: The set of jobs in order j.

OUTPUT: A shared schedule for the set of jobs in order j.

Step 1: Group the machines according to their finish times, so that the machines in each group have the same finish time. Let the groups of machines be $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_G$, so that the finish time of the machines in $\mathcal{M}_1$ is less than or equal to that of the machines in $\mathcal{M}_2$, and so on.

Step 2: Let s be the finish time of group $\mathcal{M}_1$. For each job l in order j, let $L_s(l, j) = p_{lj}$. Let $\iota = 1$.

Step 3: Reorder the machines in group $\mathcal{M}_\iota$ in descending order of their speeds.

Step 4: {*Assigning the jobs to the machines in* $\mathcal{M}_\iota$}
Let m_ι be the number of unassigned machines in group $\mathcal{M}_\iota$ at time s.
Let n_h be the number of jobs at the highest level at s.
If $n_h < m_\iota$, assign the n_h jobs to execute at the same rate on the fastest n_h machines.
Otherwise, assign the n_h jobs to execute at the same rate on the m_ι machines.
If there are any machines left, then consider the jobs at the next highest level and so on.
Continue such an assignment until a time t when one of the following events occurs:

Event 1: Some jobs are completed.

Event 2: There are two jobs l_1 and l_2 of order j such that

$$L_s(l_1, j) > L_s(l_2, j) \text{ but } L_t(l_1, j) = L_t(l_2, j).$$

Event 3: The finish time of some machines (the other machines may become idle due to some Type-1 Events occurred earlier) in $\mathcal{M}_\iota$ becomes equal to that of the machines in $\mathcal{M}_{\iota+1}$.

Set $s = t$, goto Step 5.

Step 5: {*Handling the events*}
If Event 1 occurs, keep executing the assignment on other machines (note that these machines are faster than those on which Event 1 occurs). If all jobs are finished, then goto Step 6.
If Event 2 occurs, goto Step 4.
If Event 3 occurs, delete from M_ι the machines on which the event occurs; add them into $M_{\iota+1}$; let $\iota = \iota + 1$, goto Step 3.

Step 6: Stop the procedure.

It should be noted that the above procedure actually works the same as the level algorithm due to Horvath, Lam and Sethi [31] if their algorithm is applied to schedule a set of independent jobs.

After the above procedure determines a shared schedule for the jobs of order j, the following procedure is used to construct a preemptive schedule from the portion of the shared schedule:

PROCEDURE *Interval-Preemptive-Schedule*

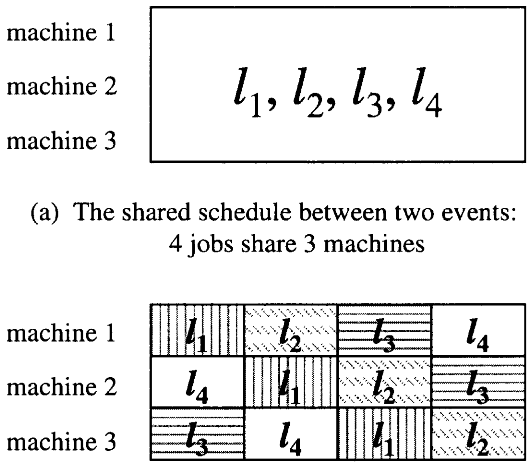
INPUT: The portion of the shared schedule.

OUTPUT: A portion of preemptive schedule.

Step 1: Divide the interval of the shared schedule into $\overline{n}$ equal subintervals along the m' shared machines, where $\overline{n}$ is the number of jobs that share the machines in this interval.

Step 2: Assign the $\overline{n}$ jobs so that each job occurs in exactly one non-overlapped subinterval on each machine.

An example of the assignment in Step 2 is shown in Figure 6.1 for $\overline{n} = 4$ and $m' = 3$.



(b) The preemptive schedule constructed from (a)

Figure 6.3 Illustrating the assignment in Step 2 of procedure *Interval-Preemptive-Schedule.*

Now with the above two procedures, it is easy to present the whole algorithm for scheduling the n orders.

ALGORITHM *Preemptive-Schedule*

INPUT: The complete set of orders to be scheduled.

OUTPUT: A complete preemptive schedule.

Step 1: Relabel the orders such that $p_1/w_1 \leq p_2/w_2 \leq \ldots \leq p_n/w_n$, where $p_j = \sum_{l=1}^{k} p_{lj}$, $j = 1, 2, \ldots, n$.

Step 2: For each order $j = 1, 2, \ldots, n$, run procedure *Shared-Schedule* on the set of jobs of order j, to produce a shared schedule. For each portion of the shared schedule, run procedure *Interval-Preemptive-Schedule* to construct the corresponding preemptive schedule for this portion.

Now consider the time complexity of the algorithm. It is easy to see that Step 1 takes $O(kn + n \log n)$ time. In Step 2, for the set of jobs of each order, there is a run of procedure *Shared-Schedule*, incorporated with a series of runs of procedure *Interval-Preemptive-Schedule.* For each order, Event 1 and Event 2 can occur at most k times each, and Event 3 can occur at most m times. In a run of *Interval-Preemptive-Schedule*, $\overline{n}$ is at most k and m' is at most m, therefore, writing down the schedule takes $O(km)$ time. Thus, each order takes at most $O(km(k+m))$ time. For all orders, the algorithm takes $O(nkm(k+m))$ time to construct the preemptive schedule. Adding the time it takes in Step 1, the overall running time of the whole algorithm is $O(nkm(k+m)+n\log n)$.

Now consider the worst-case performance ratio of the algorithm. For convenience, the algorithm *Preemptive-Schedule* is denoted by H_P. Let $\overline{C}_j$ denote the makespan obtained by applying the algorithm only to the set of jobs of each order j, assuming that each machine is available from time zero on. As mentioned before, for a set of independent jobs, H_P is essentially the same as the level algorithm due to Horvath, Lam and Sethi [31]. Thus, according to Theorem 6.1,

$$\overline{C}_j = \max\left\{\frac{p_j}{v}, \max_{1\le i<m}\left\{\frac{\sum_{l=1}^{i} p_{lj}}{\sum_{l=1}^{i} v_l}\right\}\right\}. \tag{6.8}$$

With additional constraints on the processing times of the orders, the problem may become easy. The following theorem shows that the algorithm H_P is optimal for a special case.

Theorem 6.3 *For the problem $QF \mid \Pi, prmt \mid \sum w_jC_j$, if for each order $j = 1,2,\dots,n$,*

$$\frac{p_j}{v} \ge \max_{1\le i<m}\left\{\frac{\sum_{l=1}^{i} p_{lj}}{\sum_{l=1}^{i} v_l}\right\}, \tag{6.9}$$

then H_P is optimal.

Proof: From (6.8) and (6.9),

$$\overline{C}_j = \frac{p_j}{v}.$$

Thus, it is easy to see that, after each order is assigned, all machines have the same finish time. It follows that

$$C_j(H_P) = \sum_{x=1}^{j} \frac{p_x}{v}. \tag{6.10}$$

On the other hand, for an optimal schedule, it is easy to see that

$$C_{[j]}(OPT) = \sum_{x=1}^{j} \frac{p_{[x]}}{v}. \tag{6.11}$$

Therefore, from (6.10) and (6.11)

$$\frac{\sum w_j C_j(H_P)}{\sum w_j C_j(OPT)} = \frac{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{v}}{\sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} \frac{p_{[x]}}{v}} \leq \frac{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{v}}{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{v}} = 1. \tag{6.12}$$

The inequality in (6.12) is due to the *WSPT* rule and the ordering in (6.1). The upper bound 1 in (6.12) implies that H_P has to be optimal under the additional condition constrained by (6.9); otherwise, it leads to a contradiction. □

When the number of machines m is arbitrary, it can be shown that the worst-case ratio of H_P is m.

Theorem 6.4 *For the problem $QF \mid \Pi, prmp \mid \sum w_j C_j$,*

$$\frac{\sum w_j C_j(H_P)}{\sum w_j C_j(OPT)} \leq m.$$

Proof: According to the algorithm, the sequence of orders must be

$$< 1, 2, \ldots, n > .$$

Otherwise, one can always relabel the orders to make it so.

First consider the finish time of any order $j = 1, 2, \ldots, n$. In the worst case, all the jobs of the first j orders are scheduled on machine 1 (the fastest machine), such that order j completes at time $\sum_{x=1}^{j} p_x/v_1$. Clearly, due to the preemptive behavior of the algorithm, if any portion of a job belonging to the first j orders was previously assigned to other machines, the completion time of order j would be less. Thus, it is easy to see that

$$C_j(H_P) \leq \sum_{x=1}^{j} p_x/v_1.$$

According to the assumption that

$$v_1 \geq v_2 \geq \ldots \geq v_m \text{ and } v = \sum_{i=1}^{m} v_i,$$

$$v_1 \geq \frac{v}{m}.$$

It follows that

$$C_j(H_P) \leq m \sum_{x=1}^{j} p_x/v.$$

On the other hand, it is easy to see that

$$C_{[j]}(OPT) \geq \frac{\sum_{x=1}^{j} p_{[x]}}{v}.$$

Therefore,

$$\frac{\sum w_j C_j(H_P)}{\sum w_j C_j(OPT)} \leq \frac{\sum_{j=1}^{n} w_j \cdot \left(m \sum_{x=1}^{j} p_x/v\right)}{\sum_{j=1}^{n} w_{[j]} \cdot \left(\sum_{x=1}^{j} p_{[x]}/v\right)} \leq m.$$

The last "$\leq$" in the above inequality is due to the *WSPT* rule and the ordering in (6.1). This completes the proof. $\square$

To see that H_P could perform badly, the following example is considered. Note that the example is slightly modified from the one for the non-preemptive heuristic:

- Let $n = 4$ and $w_j = 1, \;\; j = 1, 2, 3, 4$; let $k = m^2 + 1$.
- There are $m^2 + 4$ machines, with speeds: $v_1 = m, v_2 = v_3 = \ldots = v_{m^2+4} = 1$.

- The processing times of the 4 orders are:
  $p_{11} = m^2, p_{21} = p_{31} = \ldots = p_{k1} = 0;$
  $p_{1j} = 2m, p_{2j} = p_{3j} = \ldots = p_{kj} = 1$, for each $j = 2, 3, 4$.

When m is sufficiently large, it is easy to see that the schedule produced by H_P is the one shown in Figure 6.4. Thus,

$$\sum_{j=1}^{4} C_j(H_P) = m + (m+1) + (m+2) + (m+3) = 4m + 6.$$

On the other hand, if one sequences the orders as $< 2, 3, 4, 1 >$, it is easy to assign the jobs of the orders to obtain an optimal schedule in which the orders are completed on machine 1 with the following completion times:

$$\begin{aligned}
C_2 &= 2; \\
C_3 &= C_2 + \frac{2m - C_2}{m} = 4 - \frac{2}{m}; \\
C_4 &= C_3 + \frac{2m - C_3}{m} = 6 - \frac{6}{m} + \frac{2}{m^2}; \\
C_1 &= C_4 + \frac{m^2 - C_4}{m} = m + 6 - \frac{12}{m} + \frac{8}{m^2} - \frac{2}{m^3}.
\end{aligned}$$

Thus,

$$\sum_{j=1}^{4} C_j(OPT) = m + 18 - \frac{20}{m} + \frac{10}{m^2} - \frac{2}{m^3}.$$

Therefore, when $m \to \infty$,

$$\lim_{m\to\infty} \left(\frac{\sum_{j=1}^{4} C_j(H_P)}{\sum_{j=1}^{4} C_j(OPT)} \right) = \lim_{m\to\infty} \left(\frac{4m+6}{m + 18 - \frac{20}{m} + \frac{10}{m^2} - \frac{2}{m^3}} \right) = 4.$$

Again, it is easy to generalize this example and generate a series of instances for which the worst case ratio increases in $O(\sqrt{m})$.

Even though Theorem 6.4 shows that, for the general case, the worst-case performance bound of H_P is m for m machines, one might expect that the performance bound could be smaller for a small number of machine, for example, when $m = 3$. In

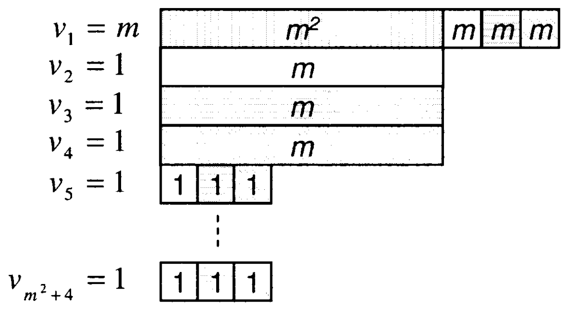


Figure 6.4 The schedule produced by H_P.

what follows, it will be shown that the bound is 2 for $m = 3$. To show this result, the following lemma is needed:

Lemma 6.5 *Given a set of independent jobs, reducing the processing time of any job will not increase the optimal schedule length.*

Proof: According to Theorem 6.1, the makespan of an optimal schedule is

$$\omega = \max\left\{\frac{\sum_{x=1}^{n} p_x}{\sum_{i=1}^{m} v_i}, \max_{1\le i<m}\left\{\frac{\sum_{x=1}^{i} p_x}{\sum_{x=1}^{i} v_x}\right\}\right\}.$$

Thus, decreasing the processing time of any job results in that ω decreases or remains unchange. □

Theorem 6.6 *For the problem $QF3 \mid \Pi, prmp \mid \sum w_jC_j$,*

$$\frac{\sum w_jC_j(H_P)}{\sum w_jC_j(OPT)} \le 2.$$

Proof: Again, according to the algorithm, the sequence of orders must be

$$< 1, 2, \dots, n > .$$

Otherwise, one can always relabel the orders to make it so.

First consider the finish time of any order $j = 1, 2, \dots, n$. For each machine $i = 1, 2, 3$, let F_i denote the finish time on machine i prior to the assignment of jobs of order j, and let F_i' denote the finish time on machine i after the jobs of order j are assigned. Clearly, $F_3 \leq F_2 \leq F_1$ always holds since $v_1 \geq v_2 \geq v_3$. Four cases are considered.

Case 1: $F_1 = F_2 = F_3$. This case is easy. It is clear that

$$C_j(H_P) = \sum_{x=1}^{j-1} p_x/v + \overline{C}_j.$$

Case 2: $F_1 = F_2 > F_3$. Two subcases are considered.

Subcase 1: $F_1' = F_2' = F_1 = F_2$ and $F_3' \leq F_1$. It is easy to see that

$$C_j(H_P) = F_3' \leq \sum_{x=1}^{j} p_x/v,$$

since there is no idle time on the three machines between 0 and F_3'.

Subcase 2: $F_1' > F_1, F_2' \geq F_1$ and $F_3' \geq F_1$. Since all machines are filled up to $F_1 = F_2$, it is easy to see that

$$F_1 = F_2 \leq \sum_{x=1}^{j} p_x/v.$$

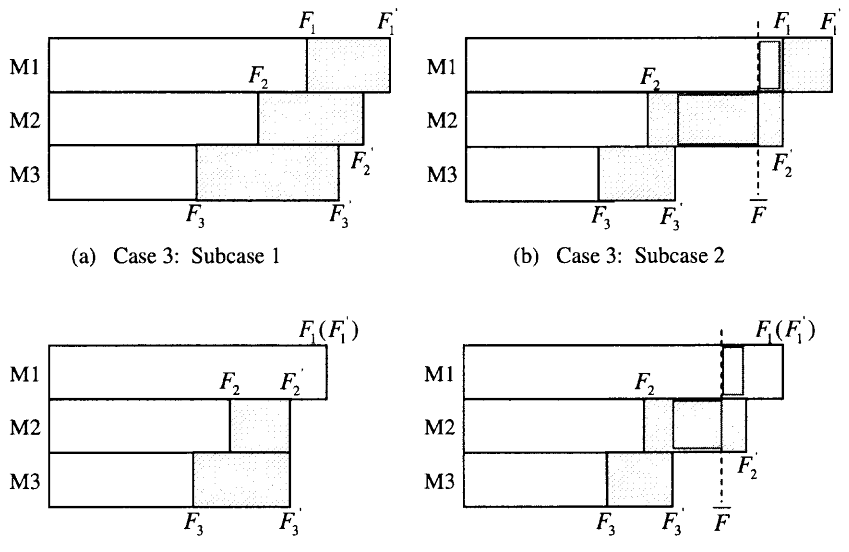
Since the unassigned portion of the jobs of order j will be assigned to the three machines starting from time point F_1, by Lemma 6.5,

$$C_j(H_P) \leq F_1 + \overline{C}_j \leq \sum_{x=1}^{j} p_x/v + \overline{C}_j.$$

Case 3: $F_1 > F_2 > F_3$. Only the possible subcases are considered.

Subcase 1: $F_1' \geq F_1$, $F_2' \geq F_1$ and $F_3' \geq F_1$, as shown in Figure 6.5 (a). It is easy to see that

$$C_j(H_P) \leq F_1 + \overline{C}_j \leq \sum_{x=1}^{j} p_x/v + \overline{C}_j.$$



(a) Case 3: Subcase 1 (b) Case 3: Subcase 2

(c) Case 3: Subcase 3 (d) Case 3: Subcase 4

Figure 6.5 Some subcases in Case 3.

Subcase 2: $F_1' \geq F_1$, $F_2' = F_1$ and $F_2 \leq F_3' < F_1$, as shown in Figure 6.5 (b). In this subcase, consider a point $\overline{F}$ such that

$$(F_1 - \overline{F}) \cdot (v_1 + v_2) = (\overline{F} - F_3') \cdot v_3.$$

Since $v_3 \leq v_2$, it follows that

$$(F_1 - \overline{F}) \cdot v_1 \leq (\overline{F} - F_3') \cdot v_2.$$

Note that after F_3', only one job could be left. Furthermore, this job must be the longest one in order j. Thus,

$$(\overline{F} - F_3') \cdot v_2 + (F_1' - F_1) \cdot v_1 \leq p_{1j}.$$

Therefore,

$$(F_1 - \overline{F}) + (F_1' - F_1) \leq \frac{p_{1j}}{v_1}.$$

In addition, since machine 3 can be filled up to $\overline{F}$ by the portion of jobs between $\overline{F}$ and F_1 on machine 1 and machine 2, it follows that

$$\overline{F} \le \frac{\sum_{x=1}^{j} p_x}{v}.$$

Thus,

$$C_j(H_P) = \overline{F} + (F_1 - \overline{F}) + (F_1' - F_1) \le \frac{\sum_{x=1}^{j} p_x}{v} + \frac{p_{1j}}{v_1} \le \sum_{x=1}^{j} p_x/v + \overline{C}_j.$$

Subcase 3: $F_1' = F_1$, $F_2 < F_2' = F_3' < F_1$, as shown in Figure 6.5 (c). It is easy to see that

$$C_j(H_P) \le \sum_{x=1}^{j} p_x/v.$$

Subcase 4: $F_1' = F_1$, $F_2 < F_2' < F_1$ and $F_2 \le F_3' < F_2'$, as shown in Figure 6.5 (d). Again, consider a point $\overline{F}$ such that

$$(F_2' - \overline{F}) \cdot (v_1 + v_2) = (\overline{F} - F_3') \cdot v_3.$$

It follows that

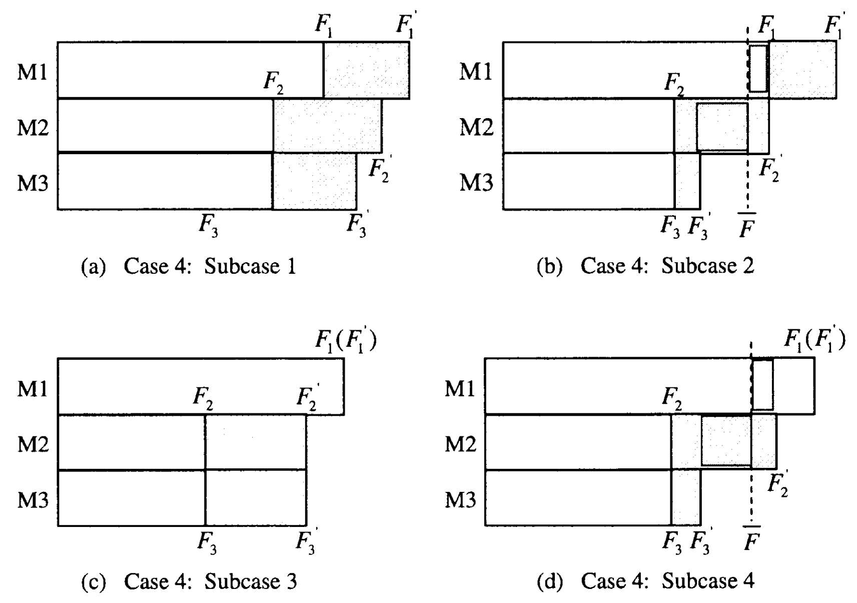
$$(F_2' - \overline{F}) \cdot v_1 \le (\overline{F} - F_3') \cdot v_3 \le (\overline{F} - F_3') \cdot v_2.$$

Note that after F_3', only one job could be left. Furthermore, this job must be the longest one in order j. Therefore,

$$(\overline{F} - F_3') \cdot v_2 \le p_{1j}.$$

Thus,

$$(F_2' - \overline{F}) \cdot v_1 \le p_{1j}.$$



(a) Case 4: Subcase 1 (b) Case 4: Subcase 2 (c) Case 4: Subcase 3 (d) Case 4: Subcase 4

Figure 6.6 The subcases in Case 4.

Again, since machine 3 can be filled up to $\overline{F}$ by the portion of jobs between $\overline{F}$ and F_2' on machine 1 and machine 2, it follows that

$$\overline{F} \leq \frac{\sum_{x=1}^{j} p_x}{v}.$$

Hence,

$$C_j(H_P) = \overline{F} + (F_2' - \overline{F}) \leq \frac{\sum_{x=1}^{j} p_x}{v} + \frac{p_{1j}}{v_1} \leq \sum_{x=1}^{j} p_x/v + \overline{C}_j.$$

Subcase 5: $F_1' = F_1$, $F_2' = F_2$ and $F_3' \leq F_2$. This subcase is easy, so it is not illustrated by a figure. As subcase 3, it is easy to see that

$$C_j(H_P) = F_3' \leq \sum_{x=1}^{j} p_x/v,$$

since there is no idle time on the three machines between 0 and F_3'.

Case 4: $F_1 > F_2 = F_3$. Again, only the possible subcases are considered.

Subcase 1: $F_1' \geq F_1$, $F_2' \geq F_1$ and $F_3' \geq F_1$. It is easy to see that

$$C_j(H_P) \leq F_1 + \overline{C}_j \leq \sum_{x=1}^{j} p_x/v + \overline{C}_j.$$

Subcase 2: $F_1' \geq F_1$, $F_2' = F_1$ and $F_2 \leq F_3' < F_1$. This is the same as Subcase 2 in Case 3. Thus,

$$C_j(H_P) \leq \sum_{x=1}^{j} p_x/v + \overline{C}_j.$$

Subcase 3: $F_1' = F_1$, $F_2 < F_2' = F_3' < F_1$. It is easy to see that

$$C_j(H_P) \leq \sum_{x=1}^{j} p_x/v.$$

Subcase 4: $F_1' = F_1$, $F_2 < F_2' < F_1$ and $F_2 \leq F_3' < F_2'$. This is the same as Subcase 4 in Case 3. Thus,

$$C_j(H_P) \leq \sum_{x=1}^{j} p_x/v + \overline{C}_j.$$

Therefore, in all cases, it satisfies

$$C_j(H_P) \leq \sum_{x=1}^{j} p_x/v + \overline{C}_j. \tag{6.13}$$

On the other hand, let the processing requirements of the two longest jobs among the first j orders scheduled in the optimal schedule be $\overline{p}_1$ and $\overline{p}_2$. It is clear that

$$\begin{aligned} C_{[j]}(OPT) &\geq \max\left\{\frac{\sum_{x=1}^{j} p_{[x]}}{\sum_{i=1}^{m} v_i}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i}\overline{p}_l}{\sum_{l=1}^{i} v_l}\right\}\right\} \\ &\geq \max\left\{\sum_{x=1}^{j}\frac{p_{[x]}}{v}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i} p_{l[j]}}{\sum_{l=1}^{i} v_l}\right\}\right\}, \end{aligned} \tag{6.14}$$

due to $\overline{p}_l \geq p_{l[j]}$ for $l = 1, 2$.

From (6.13) and (6.14)

$$\begin{aligned}
\frac{\sum w_j C_j(H_P)}{\sum w_j C_j(OPT)} &\leq \frac{\sum_{j=1}^{n} w_j \left(\sum_{x=1}^{j} \frac{p_x}{v} + \max\left\{\frac{p_j}{v}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i} p_{lj}}{\sum_{l=1}^{i} v_l}\right\}\right\}\right)}{\sum_{j=1}^{n} w_{[j]} \cdot \max\left\{\sum_{x=1}^{j} \frac{p_{[x]}}{v}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i} p_{l[j]}}{\sum_{l=1}^{i} v_l}\right\}\right\}} \\
&\leq \frac{\sum_{j=1}^{n} w_j \sum_{x=1}^{j} \frac{p_x}{v}}{\sum_{j=1}^{n} w_{[j]} \sum_{x=1}^{j} \frac{p_{[x]}}{v}} + \frac{\sum_{j=1}^{n} w_j \cdot \max\left\{\frac{p_j}{v}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i} p_{lj}}{\sum_{l=1}^{i} v_l}\right\}\right\}}{\sum_{j=1}^{n} w_{[j]} \cdot \max\left\{\sum_{x=1}^{j} \frac{p_{[x]}}{v}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i} p_{l[j]}}{\sum_{l=1}^{i} v_l}\right\}\right\}} \\
&\leq \frac{\sum_{j=1}^{n} w_j \cdot \sum_{x=1}^{j} \frac{p_x}{v}}{\sum_{j=1}^{n} w_j \cdot \sum_{x=1}^{j} \frac{p_x}{v}} + \frac{\sum_{j=1}^{n} w_j \cdot \max\left\{\frac{p_j}{v}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i} p_{lj}}{\sum_{l=1}^{i} v_l}\right\}\right\}}{\sum_{j=1}^{n} w_{[j]} \cdot \max\left\{\frac{p_{[j]}}{v}, \max_{1\leq i<3}\left\{\frac{\sum_{l=1}^{i} p_{l[j]}}{\sum_{l=1}^{i} v_l}\right\}\right\}} \\
&= 2. \qquad (6.15)
\end{aligned}$$

The first part of the last inequality in (6.15) is due to the *WSPT* rule and the ordering in (6.1). This completes the proof. □

6.4 Empirical Analysis

To evaluate the two heuristics empirically, problem instances are generated with various problem sizes that are determined by n, m and k, where $n \in \{20, 50, 100, 200\}$, $m \in \{2, 5, 10, 20\}$ and $k \in \{2, 5, 10, 20, 50, 100\}$. For each combination of n, m and k, 10 problem instances are randomly generated. These 10 problem instances have a similar structure and are treated as a group. To produce an instance for a combination of n, m and k, the speeds of the m machines are first generated. Each machine speed is generated from the uniform distribution $[1, 50]$. Then, n orders are generated. For each order j, the number of product types k_j is generated from the uniform distribution $[1, k]$. Then, for each product type $l = 1, 2, \ldots, k_j$, an integer processing time p_{lj} is generated from the uniform distribution $[1, 200]$. In addition, a weight for order j is randomly generated from the uniform distribution $[1, 20]$. In total, $4 \times 4 \times 6 \times 10 = 960$ instances are generated.

The algorithms are implemented in C++. The running environment is based on the Windows 2000 operating system; the PC used was a notebook computer

(Pentium III 900Mhz plus 384MB RAM). From experimental observation, since the time used by either heuristic to solve any problem instance costs only several milliseconds, running times are not considered. It is focused on an analysis of the performance of the algorithms. In particular, since it is unlikely that the optimal solution can be obtained by an exact algorithm very quickly, the heuristic results are compared with a lower bound of the optimal solution which can be computed easily.

From the analyses in previous sections, it is easy to see that a lower bound for both the cost of the optimal non-preemptive schedule and that of the optimal preemptive schedule is:

$$LB = \sum_{j=1}^{n} w_j \sum_{h=1}^{j} p_h / v.$$

Thus, the ratio

$$r(H) = \frac{\sum w_j C_j(H)}{LB} \geq \frac{\sum w_j C_j(H)}{\sum w_j C_j(OPT)} \geq 1,$$

where $H \in \{H_{NP}, H_P\}$. If $r(H)$ is close to 1, it means that the heuristic result is close to the lower bound. Hence, it would be even closer to the optimal cost. Therefore, to some extent, the ratio $r(H)$ indicates how good the heuristics are when they are applied to solve the problem instances.

For each instance generated, both H_{NP} and H_P are run on it to produce a non-preemptive schedule and a preemptive schedule, respectively. In addition, the value of LB is computed. With these, both $r(H_{NP})$ and $r(H_P)$ are computed for this instance. As mentioned before, for each combination of n, m, and k, the 10 problem instances that are randomly generated have a similar structure so that they can be treated as a group. It would be more reasonable to study the average ratio of the two heuristics for each group of the instances.

Table 6.1 shows the average ratio for each group of the 10 instances for each combination of n, m, and k. In the table, $\overline{r}(H_{NP})$ and $\overline{r}(H_P)$ denote the average ratio for H_{NP} and H_P, respectively. From Table 6.1, one can observe the following:

- The largest $\overline{r}(H_{NP})$ and $\overline{r}(H_P)$ are 2.113 and 1.866, respectively. This indicates that for all of the problem instances generated, the worst-case heuristic results are about 2 times the lower bound of the optimal costs. This worst case occurs for $n = 20$, $m = 20$, and $k = 2$. However, this does not mean that the heuristics perform badly for small n, k and large m. Instead, it is believed that the large ratios are caused by the gap between LB and the actual optimal cost. It is noticed that when $k = 2$, each order has at most two product types. Thus, LB would be very small if one computes p_j/v. In contrast, in the optimal schedules, the assignment of the jobs of each order may involve only a few machines and its finish time would have a closer relationship to $\max\{p_{lj}\}$ than p_j/v. Therefore, computing LB may underestimate the optimal cost too much, which results in large values of $\overline{r}(H_{NP})$ and $\overline{r}(H_P)$.

- For each combination of n and m, $\overline{r}(H)$ drops close to 1.0 when k grows from 2 to 100. This indicates that the schedules produced by H_{NP} and H_P are very close to the lower bound of the optimal schedules. Hence, they are even closer to the optimal schedules. The reason may lie in that, when k is large, each order may have a lot of different product types to produce so that the data of the orders are more regular for the heuristics to obtain good schedules.

- For each combination of n and k, $\overline{r}(H)$ grows when m becomes large. This indicates that the heuristics perform better when the number of machines is smaller. The observation is consistent with the worst-case bounds that have been shown for the two heuristics in previous sections.

- For each combination of m and k, $\overline{r}(H)$ drops when n becomes large. However, it is not concluded that the heuristics may perform better for a large number of orders than for a small number. Again, it is believed that this may be caused by the gap between the lower bound and the optimal cost.

From the above observations, it is quite clear that the two heuristics can produce very near-optimal solutions for the randomly generated instances with large k and small m.

Table 6.1 The Average Ratio of H_{NP} and H_P over the Instances Generated for Each Combination of n, m, and k

		$m = 2$		$m = 5$		$m = 10$		$m = 20$	
n	k	$\bar{r}(H_{NP})$	$\bar{r}(H_P)$	$\bar{r}(H_{NP})$	$\bar{r}(H_P)$	$\bar{r}(H_{NP})$	$\bar{r}(H_P)$	$\bar{r}(H_{NP})$	$\bar{r}(H_P)$
20	2	1.052	1.029	1.227	1.163	1.541	1.417	2.113	1.866
	5	1.024	1.010	1.101	1.053	1.303	1.208	1.726	1.563
	10	1.009	1.003	1.043	1.013	1.141	1.079	1.413	1.290
	20	1.003	1.000	1.019	1.006	1.057	1.024	1.182	1.116
	50	1.001	1.000	1.004	1.001	1.014	1.005	1.038	1.018
	100	1.000	1.000	1.002	1.000	1.004	1.001	1.015	1.006
50	2	1.024	1.011	1.095	1.062	1.228	1.174	1.491	1.364
	5	1.009	1.003	1.052	1.030	1.123	1.081	1.304	1.219
	10	1.004	1.001	1.020	1.006	1.062	1.036	1.162	1.114
	20	1.001	1.000	1.009	1.002	1.022	1.008	1.076	1.050
	50	1.000	1.000	1.002	1.000	1.005	1.001	1.016	1.007
	100	1.000	1.000	1.000	1.000	1.002	1.000	1.006	1.002
100	2	1.011	1.005	1.047	1.029	1.114	1.080	1.244	1.178
	5	1.004	1.001	1.026	1.016	1.060	1.041	1.146	1.104
	10	1.002	1.000	1.010	1.004	1.030	1.018	1.081	1.058
	20	1.001	1.000	1.004	1.001	1.013	1.005	1.035	1.021
	50	1.000	1.000	1.001	1.000	1.003	1.001	1.009	1.004
	100	1.000	1.000	1.000	1.000	1.001	1.000	1.003	1.001
200	2	1.005	1.002	1.025	1.016	1.056	1.040	1.120	1.086
	5	1.002	1.001	1.013	1.008	1.031	1.021	1.069	1.048
	10	1.001	1.000	1.006	1.002	1.016	1.009	1.040	1.026
	20	1.000	1.000	1.002	1.001	1.006	1.003	1.018	1.010
	50	1.000	1.000	1.000	1.000	1.002	1.000	1.005	1.002
	100	1.000	1.000	1.000	1.000	1.000	1.000	1.001	1.000

6.5 Concluding Remarks

This chapter proposed two approximation algorithms for order scheduling on uniform machines, one for non-preemptive scheduling (H_{NP}) and the other for preemptive scheduling (H_P). It was shown that the worst-case bound for both heuristics is m. Most probably, the worst-case bounds for both H_{NP} and H_P are not tight, even though examples could be constructed to show that they perform badly. Since the counterexamples show that the two heuristics are not bounded by any constant, it would be of interest to investigate whether the heuristics are bounded by a sub-linear function of m, for example, $O(\sqrt{m})$.

The two heuristics were also implemented to have an empirical analysis. The observations on the experimental results reveal that the two heuristics can produce in practice solutions that are very close to optimal. Due to their solution qualities and fast speeds, the two heuristics are recommended to solve large-sized real-life problem instances.

CHAPTER 7

CONCLUSIONS

In this dissertation, a variety of problems that belong to order scheduling models were studied. Attention was focused on some problems of the fully dedicated case and of the fully flexible case. Complexity results were established for some of the problems that are either polynomially solvable or NP-hard. For the NP-hard problems, various simple heuristics were designed for each problem, and worst-case analyses as well as empirical analyses were presented for these heuristics.

In the complexity aspect, Table 7.1 summarizes the complexity status of order scheduling problems that were studied in the dissertation and in the literature. From this table, it can be seen that a lot of problems remain open. For example, for those problems that are ordinary NP-hard, are there any pseudo-polynomial time algorithms for them, or are they NP-hard in the strong sense?

In the algorithm aspect, the algorithms that were studied in this dissertation for the NP-hard order scheduling problems can be categorized into five groups:

Exact Algorithm: An algorithm of this group is expected to return an optimal solution for an NP-hard problem. However, the exact algorithm usually takes a very long time (possibly, millions of years) to return. It is not affordable to wait for such a long time. Usually, elimination rules and constraint propagation techniques can be used to make an exact algorithm run faster. Unfortunately, even with such techniques, the running time of the exact algorithm is still unpredictable. In Chapter 3, the exact algorithm for solving $PD \mid\mid \sum U_j$ belongs to this category.

Constant Ratio Approximation Algorithm: An algorithm of this group returns a quality-guaranteed solution for a problem in polynomial time. The objective cost of the solution is at most a constant times the optimal cost. Examples of such algorithms are the five *WSTP*-based heuristics in Chapter 5 for the problem $PF \mid \Pi \mid \sum w_j C_j$, and the two LP-based approximation algorithms in Chapter 4 for the problem $PD \mid\mid \sum w_j C_j$.

Table 7.1 The Complexity Status of the Order Scheduling Problems Studied in the Dissertation and in the Literature

Problem	Complexity Status
$PDm \mid\mid \sum C_j$	Strongly NP-hard for each $m \geq 3$ (Chapter 2); remains open for $m = 2$
$PDm \mid\mid \sum w_jC_j$	Strongly NP-hard for each $m \geq 2$ [63]
$PD \mid prec \mid f_{\max}$	Polynomially solvable (Chapter 3)
$PD \mid\mid L_{\max}$	Polynomially solvable (Chapter 3)
$PD \mid r_j, prmp \mid L_{\max}$	Polynomially solvable (Chapter 3)
$PD \mid \bar{d}_j \mid L_{\max}$	Polynomially solvable (Chapter 3)
$PD2 \mid\mid \sum U_j$	Ordinary NP-hard [65]; pseudo-polynomial algorithm [10]
$PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$	Strongly NP-hard [55]
$PF2 \mid \Pi \mid \sum C_j$	Ordinary NP-hard [4, 67]; remains open for pseudo-polynomial algorithms
$PF \mid \Pi \mid \sum C_j$	Strongly NP-hard [4]
$PF2 \mid \Pi 2, prmp \mid \sum C_j$	Ordinary NP-hard [44]; remains open for pseudo-polynomial algorithms
$QF \mid \Pi \mid \sum C_j$	Strongly NP-hard, generalization of $Q \mid\mid C_{max}$
$QF \mid \Pi k \mid \sum w_jC_j$	Strongly NP-hard, generalization of $P \mid\mid \sum w_jC_j$
$PF1 \mid s_l, \Pi \mid L_{max}$	Ordinary NP-hard and pseudo-polynomial algorithm [20]
$PF1 \mid s_l, \Pi, GT \mid L_{max}$	Polynomial algorithm [20]
$PF1 \mid s_l, \Pi k \mid \sum U_j$	Ordinary NP-hard and pseudo-polynomial algorithm [20]
$PF1 \mid s = 1, \Pi \mid \sum U_j$	Strongly NP-hard [20, 9]
$PF1 \mid s, \Pi, p_{lj} \in \{0,1\} \mid \sum C_j$	Strongly NP-hard [54]; remains open for fixed k
$PF1 \mid s, \Pi, GT, p_{lj} \in \{0,1\} \mid \sum C_j$	Strongly NP-hard [54]; remains open for fixed k

Table 7.2 m-Approximation Algorithms in the Dissertation

Chapter	Problem	Heuristics
2	$PD \mid\mid \sum C_j$	*STPT*, *SMPT*, *SMCT*, *ECT*
4	$PD \mid\mid \sum w_j C_j$	*WSTP*, *WSMP*, *WSMC*, *WECT*
6	$QF \mid \Pi \mid \sum w_j C_j$	H_{NP}
6	$QF \mid \Pi, prmp \mid \sum w_j C_j$	H_P

Algorithm with a Worst-Case Ratio of Logarithmic Function: An algorithm of this group also returns a quality-guaranteed solution for the problem in polynomial time. However, the performance ratio is bounded by a logarithmic function of the problem size. An example of such algorithms is the greedy algorithm in Chapter 3 for the problem $PD \mid d_j = d \mid \sum w_j C_j$.

Algorithm with a Worst-Case Ratio of Linear Function: An algorithm of this group still returns a quality-guaranteed solution for a problem in polynomial time. However, the worst-case ratio is bounded by a linear function of the problem size. Examples of such algorithms are listed in Table 7.2.

Unbounded Algorithm: An algorithm of this group returns a solution for a problem in polynomial time, but the quality of the solution is not guaranteed by any performance ratio. In other words, the worst-case ratio of such an algorithm could be infinitely large. An example of such algorithms is the *SPTL* heuristic in Chapter 2 for the problem $PD \mid\mid \sum C_j$.

From the above discussion, one can see that the order scheduling models still have a lot of open problems about complexity and approximation. Thus, as future research, it would be interesting to work on the open problems. For those problems that have approximation algorithms, it would be interesting to investigate if the performance bounds of the algorithms can be tighter, or to investigate if the problems have polynomial-time approximation schemes (PTAS), or to show that the problems are APX-hard if such PTAS does not exist unless $\mathcal{P} = \mathcal{NP}$. Such approximability topics were addressed in Arora [2] and Papadimitriou [56].

In this dissertation, the issue of setup times was not considered. With setup times, the scheduling problems usually become considerably harder, even for a single machine. For example, some single-machine cases were studied in [20, 9, 54]. The results in the literature showed that even the very restricted problems are strongly NP-hard for certain objective functions such as $\sum U_j$ and $\sum C_j$. As shown in Table 7.1, the complexity of some problems involving setup times still remains open. Therefore, it would be of interest to study these open problems. For the NP-hard cases, it would also be worthwhile to design some efficient heuristics.

The order scheduling problems that were studied in the dissertation are very basic ones. With different machine environments, side constraints, and objectives, the problems may become more complicated. These could become research interests in the future. In addition, it would also be interesting to extend the order scheduling models to online scheduling and stochastic scheduling.

APPENDIX A

PROOF OF LEMMA 2.6

This appendix presents the proof of Lemma 2.6 in Chapter 2.

For ease of presentation, the following notation is defined to be used throughout the proof:

$F_j^{(i)}$: The length on machine $i (i = 1, 2, 3)$ after the j^{th} order is scheduled.

$C_j^{(i)}$: The finish time on machine $i (i = 1, 2, 3)$ of the j^{th} order.

$C_j = \max_{1 \leq i \leq 3}\{C_j^{(i)}\}$: The finish time of the j^{th} order in the schedule.

The above notation is defined in terms of the position of an order. In contrast, the following notation is defined in terms of the order itself:

$C_{J_j}^{(i)}$: The finish time on machine $i (i = 1, 2, 3)$ of order J_j.

$C_{J_j} = \max_{1 \leq i \leq 3}\{C_{J_j}^{(i)}\}$: The finish time of order J_j in the schedule.

In addition, for convenience of describing a pattern of a schedule, the following notation is defined:

$(P)^k$: A pattern produced by repeating k times the subpattern P, which is a sequence of a, b and c.

$(P_1/P_2)^k$: A pattern produced by repeating k times the subpattern P_1 or P_2, each of which is a sequence of a, b and c.

Furthermore, let S_{OPT} denoted an optimal schedule and let $\sum C_j(S)$ denote the total completion time of schedule S.

Finally, for convenience, for each order constructed from a_j (b_j, c_j), it is simply called order a_j (respectively, b_j, c_j). Also, it is assumed that the subscripts of the orders in a schedule are labeled in such a way that,

a_j $(b_j, \text{or } c_j)$ $(j = 1, 2, \ldots, n)$: means that there are $(j-1)$ orders of type a scheduled before a_j (respectively, $b_j, \text{or } c_j$).

Observation A.1 *In S_{OPT}:*

i) The orders of type c must be scheduled by a sequence satisfying the SPT rule;

ii) The first order in the schedule cannot be of type c;

iii) The last order in the schedule must be of type c.

Proof: *i)* It follows by an interchange argument.

ii) Suppose that S_{OPT} starts with an order of type c. From *i*), this order must be c_1. Let the first type-a or type-b order be in position $(k+1) > 1$. Denote this order as c^*. Then, the first $(k+1)$ orders are scheduled as:

$$c_1 c_2 \ldots c_k c^*.$$

The finish time of order $c_j (j = 1, 2, \ldots, k)$ in $c_1 c_2 \ldots c_k c^*$ is:

$$C_{c_j} = \max \left\{ \begin{array}{c} jL, \\ 2jL + 2\sum_{l=1}^{j} c_l, \\ j(5L+2X) + \sum_{l=1}^{j} c_l \end{array} \right\} = j(5L+2X) + \sum_{l=1}^{j} c_l,$$

If c^* is scheduled before c_1 (i.e., $c^* c_1 c_2 \ldots c_k$), it can be shown that the finish time of order $c_j (j = 1, 2, \ldots, k)$ remains the same as before. But the finish time of c^* in $c^* c_1 c_2 \ldots c_k$ is reduced by an amount of kL on machine 1 and an amount of $2kL + 2\sum_{l=1}^{k} c_l$ on machine 2. Therefore, $c^* c_1 c_2 \ldots c_k$ has smaller $\sum C_j$ than $c_1 c_2 \ldots c_k c^*$, contradicting the fact that S_{OPT} is optimal.

iii) Suppose that S_{OPT} does not end with the last type-c order c_n. If order c_n is moved to the end and push all orders scheduled after c_n forward, then the finish time of c_n in the new schedule is the same as the finish time of c_n in the old schedule. But the finish times of all orders that were pushed forward will decrease in the new schedule. Thus, the new schedule has a smaller $\sum C_j$ than S_{OPT}, contradicting the fact that S_{OPT} is optimal. □

Observation A.1 reveals the fact that it would be advantageous to distribute the remaining orders of type a and b within the n frames imposed by the orders of type c.

Observation A.2 *In S_{OPT} the first order must be of type a.*

Proof: Let $(k+1) > 1$ be the smallest position of a type-a order in S_{OPT}. Following the convention, let a_1 denote this type-a order. It will be shown that there is a schedule S which has a_1 in a position less than $(k+1)$, such that

$$\sum_{j=1}^{3n} C_j(S) < \sum_{j=1}^{3n} C_j(S_{OPT}).$$

The first k orders are either of type-b or type-c. Let the number of type-b and type-c orders be n_b and n_c, respectively. Clearly, $n_b + n_c = k$. From Observation A.1, the first order cannot be of type c. Thus, $n_b \geq 1$. Now, it needs to be shown that $n_c \geq 1$.

Suppose $n_c = 0$. Consider the schedule S obtained from S_{OPT} by interchanging a_1 with the order before it (which is b_k). Then,

$$\begin{aligned}
\sum C_j(S) - \sum C_j(S_{OPT}) &= C_{a_1}(S) - C_{b_k}(S_{OPT}) \\
&= \max\left\{\begin{array}{c} (k-1)(2L+2X) + \sum_{j=1}^{k-1} b_j + (2L + a_1), \\ (k-1)(L+X) - \sum_{j=1}^{k-1} b_j + (2L + X - a_1), \\ 0 \end{array}\right\} - \\
&\quad \max\left\{\begin{array}{c} k(2L+2X) + \sum_{j=1}^{k} b_j, \\ k(L+X) - \sum_{j=1}^{k} b_j, \\ 0 \end{array}\right\} \\
&= -2X + a_1 - b_k < 0,
\end{aligned}$$

contradicting the fact that S_{OPT} is optimal. Thus, $n_c \geq 1$.

Since $k = n_b + n_c$, it follows that $k \geq 2$. If $k = 2$, then the first three jobs of S_{OPT} must be $b_1 c_1 a_1$. Consider the schedule S obtained from S_{OPT} by swapping b_1

with a_1. It is clear that the finish time of c_1 in S remains unchanged. Thus,

$$\sum C_j(S) - \sum C_j(S_{OPT}) = C_{a_1}(S) - C_{b_1}(S_{OPT}) = -X - a_1 - b_1 < 0.$$

Therefore, S is better than S_{OPT}, contradicting the fact that S_{OPT} is optimal.

From now on, it will be assumed that $k \geq 3$. First, it holds that

$$F_{k+1}^{(1)} = n_b(2L + 2X) + n_c L + \sum_{j=1}^{n_b} b_j + (2L + a_1). \tag{A.1}$$

$$F_{k+1}^{(2)} = n_b(L + X) + 2n_c L + 2\sum_{j=1}^{n_c} c_j - \sum_{j=1}^{n_b} b_j + (2L + X - a_1). \tag{A.2}$$

$$F_{k+1}^{(3)} = n_c(5L + 2X) + \sum_{j=1}^{n_c} c_j. \tag{A.3}$$

<u>**Case 1**</u>: $F_{k+1}^{(1)} \geq F_{k+1}^{(2)}$ and $F_{k+1}^{(1)} \geq F_{k+1}^{(3)}$.

From (A.1) and (A.2),

$$F_{k+1}^{(1)} \geq F_{k+1}^{(2)} \Rightarrow n_c \leq n_b. \tag{A.4}$$

From (A.1) and (A.3),

$$F_{k+1}^{(1)} \geq F_{k+1}^{(3)} \Rightarrow n_b \geq 2n_c - 1. \tag{A.5}$$

Since $n_c \geq 1$, $n_b \geq 2n_c - 1 \Rightarrow n_c \leq n_b$. Therefore, it is necessary to focus only on $n_b \geq 2n_c - 1$. Now,

$$F_k^{(1)} = n_b(2L + 2X) + n_c L + \sum_{j=1}^{n_b} b_j, \tag{A.6}$$

$$F_k^{(2)} = n_b(L + X) + 2n_c L + 2\sum_{j=1}^{n_c} c_j - \sum_{j=1}^{n_b} b_j. \tag{A.7}$$

Thus,

$$\begin{aligned} F_k^{(1)} - F_k^{(2)} &= (n_b - n_c)L + (n_b - 1)X + \left(X - 2\sum_{j=1}^{n_c} c_j\right) + 2\sum_{j=1}^{n_b} b_j \\ &> (n_b - n_c)L + (n_b - 1)X, \qquad \text{(A.8)} \end{aligned}$$

due to $L > nX > 2n\sum_{j=1}^{n} c_j = 2n\sum_{j=1}^{n}(a_j + b_j)$. There are the following three cases:

Case 1(a): $n_c = 1$

Since $k \geq 3$ and $n_c = 1$, it has to be that $n_b \geq 2$. It can be shown that the first k orders in S_{OPT} must be of the pattern

$$S_{OPT} : b_1 b_2 c_1 (b)^{n_b - 2}.$$

Any schedules that put c_1 in positions other than the above can be converted into another one with smaller $\sum C_j$, contradicting the fact that S_{OPT} is optimal.

Consider the first $(k+1)$ orders in S_{OPT}. If $n_b = 2$, then

$$S_{OPT} : b_1 b_2 c_1 a_1.$$

Now consider

$$S : a_1 b_2 c_1 b_1.$$

It is easy to see that

$$C_{b_1}(S_{OPT}) = \max\{2L + 2X + b_1, L + X - b_1, 0\} = 2L + 2X + b_1,$$

$$C_{b_2}(S_{OPT}) = \max\{4L + 4X + b_1 + b_2, 2L + 2X - b_1 - b_2, 0\} = 4L + 4X + b_1 + b_2,$$

$$C_{c_1}(S_{OPT}) = \max\left\{\begin{array}{c} 5L + 4X + b_1 + b_2, \\ 4L + 2X + 2c_1 - b_1 - b_2, \\ 5L + 2X + c_1 \end{array}\right\} = 5L + 4X + b_1 + b_2.$$

and

$$C_{a_1}(S) = \max\{2L + a_1, 2L + X - a_1, 0\} = 2L + X - a_1,$$

$$C_{b_2}(S) = \max\{4L + 2X + a_1 + b_2, 3L + 2X - a_1 - b_2, 0\} = 4L + 2X + a_1 + b_2,$$

$$C_{c_1}(S) = \max \left\{ \begin{array}{c} 5L + 2X + a_1 + b_2, \\ 5L + 2X + 2c_1 - a_1 - b_2, \\ 5L + 2X + c_1 \end{array} \right\} < 5L + 2X + 2c_1 + a_1 + b_2.$$

Therefore,

$$\begin{aligned} & \sum C_j(S) - \sum C_j(S_{OPT}) \\ = \ & (C_{a_1}(S) + C_{b_2}(S) + C_{c_1}(S)) - (C_{b_1}(S_{OPT}) + C_{b_2}(S_{OPT}) + C_{c_1}(S_{OPT})) \\ < \ & (11L + 5X + a_1 + 2b_2 + 2c_1) - (11L + 10X + 3b_1 + 2b_2) \\ = \ & -5X + a_1 + 2c_1 - 3b_1 < 0. \end{aligned}$$

Thus, S is better than S_{OPT}.

If $n_b \geq 3$, then

$$S_{OPT} : b_1 b_2 c_1 (b)^{n_b - 3} b_{n_b} a_1.$$

It can be shown that

$$S : b_1 b_2 c_1 (b)^{n_b - 3} a_1 b_{n_b}$$

is better than S_{OPT} by:

$$\begin{aligned} & \sum C_j(S) - \sum C_j(S_{OPT}) = C_{a_1}(S) - C_{b_{n_b}}(S_{OPT}) \\ = \ & \left((n_b - 1)(2L + 2X) + \sum_{j=1}^{n_b - 1} b_j + L + (2L + a_1) \right) - \left(n_b(2L + 2X) + \sum_{j=1}^{n_b} b_j + L \right) \\ = \ & -2X - b_{n_b} + a_1 < 0. \end{aligned}$$

Therefore, for Case 1(a), one can always find a schedule better than S_{OPT}.

Case 1(b): $n_c = 2$

Since $n_b \geq 2n_c - 1$, it must have $n_b \geq 3$. By the same argument as in Case 1(a), it can be shown that the first k orders in S_{OPT} must be scheduled in either

$$b_1 b_2 c_1 b_3 c_2 \ (n_b = 3),$$

or

$$b_1 b_2 c_1 b_3 b_4 c_2 (b)^{n_b - 4} \ (n_b \geq 4).$$

In the former case, the schedule for the first $(k+1)$ orders is:

$$S_{OPT} : b_1 b_2 c_1 b_3 c_2 a_1.$$

It can be shown that

$$S : b_1 b_2 c_1 a_1 c_2 b_3$$

is better than S_{OPT} by:

$$\begin{aligned} & \sum C_j(S) - \sum C_j(S_{OPT}) \\ = & \ (C_{a_1}(S) + C_{c_2}(S)) - (C_{b_3}(S_{OPT}) + C_{c_2}(S_{OPT})) \\ = & \ -2X + a_1 - b_3 < 0. \end{aligned}$$

In the latter case, if $n_b > 4$, the schedule for the first $(k+1)$ orders is:

$$S_{OPT} : b_1 b_2 c_1 b_3 b_4 c_2 (b)^{n_b - 5} b_{n_b} a_1.$$

It can be shown that

$$S : b_1 b_2 c_1 b_3 b_4 c_2 (b)^{n_b - 5} a_1 b_{n_b}$$

is better than S_{OPT} by:

$$\begin{aligned}
\sum C_j(S) - \sum C_j(S_{OPT}) &= C_{a_1}(S) - C_{b_{n_b}}(S_{OPT}) \\
&= \left((n_b-1)(2L+2X) + \sum_{j=1}^{n_b-1} b_j + 2L + (2L+a_1)\right) - \\
&\quad \left(n_b(2L+2X) + \sum_{j=1}^{n_b} b_j + 2L\right) \\
&= -2X + a_1 - b_{n_b} < 0,
\end{aligned}$$

due to $n_b > 4$.

On the other hand, if $n_b = 4$, the schedule for the first $(k+1)$ orders is:

$$S_{OPT} : b_1 b_2 c_1 b_3 b_4 c_2 a_1.$$

It can be shown that

$$S : b_1 b_2 c_1 a_1 b_4 c_2 b_3$$

is better than S_{OPT} by:

$$\begin{aligned}
&\sum C_j(S) - \sum C_j(S_{OPT}) \\
&= (C_{a_1}(S) + C_{b_4}(S) + C_{c_2}(S)) - (C_{b_3}(S_{OPT}) + C_{b_4}(S_{OPT}) + C_{c_2}(S_{OPT})) \\
&= -3(2X - a_1 + b_3) < 0.
\end{aligned}$$

Therefore, for Case 1(b), one can always find a better schedule than S_{OPT} .

Case 1(c): $n_c \geq 3$

Since $n_b \geq 2n_c - 1$, it must have $n_b \geq 5$. From (A.8) and (A.5),

$$\begin{aligned}
F_k^{(1)} - F_k^{(2)} &> (n_b - n_c)L + (n_b - 1)X \\
&\geq (2n_c - 1 - n_c)L + 4X = (n_c - 1)L + 4X \geq 2L + 4X. \quad \text{(A.9)}
\end{aligned}$$

If the k^{th} order is of type-b, i.e., the schedule for the first $(k+1)$ orders is:

$$(b/c)^{k-1} b_{n_b} a_1,$$

then,

$$\begin{aligned}
&\left(F_{k-1}^{(1)} + (2L + 2X + b_{n_b})\right) - \left(F_{k-1}^{(2)} + (L + X - b_{n_b})\right) > 2L + 4X \\
&\Rightarrow \quad \left(\left(F_{k-1}^{(1)} + 2L + 2X + b_{n_b}\right) + 2L + a_1\right) - \\
&\qquad \left(\left(F_{k-1}^{(2)} + L + X - b_{n_b}\right) + 2L + X - a_1\right) > \\
&\qquad 2L + 4X + (2L + a_1) - (2L + X - a_1) \\
&\Rightarrow \quad \left(F_{k-1}^{(1)} + 2L + a_1\right) - \left(F_{k-1}^{(2)} + 2L + X - a_1\right) > \\
&\qquad 2L + 4X + (2L + a_1) - (2L + X - a_1) - (2L + 2X + b_{n_b}) + (L + X - b_{n_b}) \\
&\qquad = L + 2X + 2a_1 - 2b_{n_b} > L + X. \qquad \text{(A.10)}
\end{aligned}$$

This implies that, in the sequence $(b/c)^{k-1} a_1 b_{n_b}$, a_1 will still finish on machine 1. However, $(b/c)^{k-1} a_1 b_{n_b}$ has a cost reduction by an amount of $(2X + b_{n_b} - a_1)$ from $(b/c)^{k-1} b_{n_b} a_1$. Thus, this will result in a better schedule.

If the k^{th} order is of type-c, let b_{n_b} be in the $(k-l)^{th}$ $(l \geq 1)$ position, i.e., the pattern is:

$$(b/c)^{k-l-1} b_{n_b} (c)^l a_1.$$

It follows that,

$$\begin{aligned}
&\left(F^{(1)}_{k-l-1} + (2L + 2X + b_{n_b}) + lL\right) - \\
&\qquad \left(F^{(2)}_{k-l-1} + (L + X - b_{n_b}) + 2lL + 2\sum_{j=n_c-l+1}^{n_c} c_j\right) > 2L + 4X \\
&\Rightarrow \left(\left(F^{(1)}_{k-l-1} + 2L + 2X + b_{n_b} + lL\right) + 2L + a_1\right) - \\
&\qquad \left(\left(F^{(2)}_{k-1} + L + X - b_{n_b} + 2lL + 2\sum_{j=n_c-l+1}^{n_c} c_j\right) + 2L + X - a_1\right) > \\
&\qquad 2L + 4X + (2L + a_1) - (2L + X - a_1) \\
&\Rightarrow \left(F^{(1)}_{k-l-1} + 2L + a_1\right) - \left(F^{(2)}_{k-1} + 2L + X - a_1\right) > \\
&\qquad 2L + 4X + (2L + a_1) - (2L + X - a_1) - (2L + 2X + b_{n_b} + lL) + \\
&\qquad \left(L + X - b_{n_b} + 2lL + 2\sum_{j=n_c-l+1}^{n_c} c_j\right) \\
&\qquad = (l + 1)L + 2X + 2a_1 - 2b_{n_b} + 2\sum_{j=n_c-l+1}^{n_c} c_j > (l + 1)L + X. \qquad \text{(A.11)}
\end{aligned}$$

This implies that if one swaps a_1 with b_{n_b}, a_1 will still finish on machine 1. But it results in a cost reduction by an amount of $(2X + b_{n_b} - a_1)$. In addition, for any type-c order in position $(k - l + r)$ $(r = 1, 2, \ldots, l)$,

$$\begin{aligned}
&\left(F^{(1)}_{k-l-1} + (2L + 2X + b_{n_b}) + rL + (l - r)L\right) - \\
&\qquad \left(F^{(2)}_{k-l-1} + (L + X - b_{n_b}) + 2rL + 2\sum_{j=n_c-l+1}^{n_c-l+r} c_j + 2(l - r)L + 2\sum_{j=n_c-l+r+1}^{n_c} c_j\right) \\
&\qquad > 2L + 4X \\
&\Rightarrow \left(F^{(1)}_{k-l-1} + (2L + 2X + b_{n_b}) + rL\right) - \\
&\qquad \left(F^{(2)}_{k-l-1} + (L + X - b_{n_b}) + 2rL + 2\sum_{j=n_c-l+1}^{n_c-l+r} c_j\right) \\
&\qquad > 2L + 4X - (l - r)L + 2(l - r)L + 2\sum_{j=n_c-l+r+1}^{n_c} c_j \\
&\qquad = 2L + 4X + (l - r)L + 2\sum_{j=n_c-l+r+1}^{n_c} c_j \geq 2L + 4X. \qquad \text{(A.12)}
\end{aligned}$$

Therefore, before the swap, for any position $(k-l+r)$ $(r = 1, 2, \ldots, l)$:

$$F^{(1)}_{k-l+r} - F^{(2)}_{k-l+r} > 2L + 4X.$$

This means that the type-c order in position $(k-l+r)$ finishes on machine 1 if $F^{(1)}_{k-l+r} - F^{(3)}_{k-l+r} \geq 0$; otherwise it finishes on machine 3. Furthermore, since

$$\begin{aligned}
&\left(F^{(1)}_{k-l-1} + (2L + 2X + b_{n_b}) + rL + (l-r)L\right) - \\
&\quad \left(F^{(2)}_{k-l-1} + (L + X - b_{n_b}) + 2rL + 2\sum_{j=n_c-l}^{n_c-l+r} c_j + 2(l-r)L + 2\sum_{j=n_c-l+r+1}^{n_c} c_j\right) \\
&\quad > 2L + 4X \\
\Rightarrow\ &\left(F^{(1)}_{k-l-1} + (2L + a_1) + rL\right) - \left(F^{(2)}_{k-l-1} + (2L + X - a_1) + 2rL + 2\sum_{j=n_c-l}^{n_c-l+r} c_j\right) \\
&\quad > 2L + 4X - (2L + 2X + b_{n_b}) + (2L + a_1) + (L + X - b_{n_b}) - (2L + X - a_1) \\
&\qquad -(l-r)L + 2(l-r)L + 2\sum_{j=n_c-l+r+1}^{n_c} c_j \\
&\quad = L + (l-r)L + 2X + 2a_1 - 2b_{n_b} + 2\sum_{j=n_c-l+r+1}^{n_c} c_j > L + X,
\end{aligned} \tag{A.13}$$

after the swap, for any position $(k-l+r)$ $(r = 1, 2, \ldots, l)$:

$$F^{(1)}_{k-l+r} - F^{(2)}_{k-l+r} > L + X.$$

This means that the type-c order in position $(k-l+r)$ still finishes on machine 1 if $F^{(1)}_{k-l+r} - F^{(3)}_{k-l+r} \geq 0$; otherwise it finishes on machine 3. However, since $F^{(1)}_{k-l+r}$ was reduced by an amount of $(2X + b_{n_b} - a_1)$ after the swap, the finish time of the type-c order in position $(k-l+r)$ will either remain unchanged or be reduced.

The above argument shows that swapping a_1 with b_{n_b} will result in a better schedule.

Case 2: $F^{(2)}_{k+1} \geq F^{(1)}_{k+1}$ and $F^{(2)}_{k+1} \geq F^{(3)}_{k+1}$.

From (A.2) and (A.1),

$$F^{(1)}_{k+1} \leq F^{(2)}_{k+1} \Rightarrow n_b \leq n_c. \tag{A.14}$$

From (A.2) and (A.3),

$$F_{k+1}^{(3)} \leq F_{k+1}^{(2)} \Rightarrow n_c \leq \frac{n_b + 2}{3} \Rightarrow n_c \leq \frac{n_c + 2}{3} \Rightarrow n_c \leq 1, \tag{A.15}$$

due to (A.14). Thus,

$$(k = n_b + n_c) \wedge (n_b \leq n_c) \wedge (n_c \leq 1) \Rightarrow k \leq 2.$$

However, it has been proved the case $k \leq 2$ before. So this case needs not be considered any further.

<u>**Case 3**</u>: $F_{k+1}^{(3)} \geq F_{k+1}^{(1)}$ and $F_{k+1}^{(3)} \geq F_{k+1}^{(2)}$.

In this case, the schedule must be of the pattern:

$$b_1(b/c)^l c_{n_c}(b)^{k-l-2}a_1.$$

Consider the following pattern:

$$b_1(b/c)^l(b)^{k-l-2}a_1c_{n_c}.$$

It is clear that the finish time of each order in $(b)^{k-l-2}a_1$ will be reduced by an amount of L on machine 1 and an amount of $(2L + 2c_{n_c})$ on machine 2. The finish time of order c_{n_c} remains unchanged in its new position since it finishes on machine 3. Thus, $b_1(b/c)^l(b)^{k-l-2}a_1c_{n_c}$ is better than $b_1(b/c)^l c_{n_c}(b)^{k-l-2}a_1$.

Summarizing the above cases, the correctness of the observation follows. □

Observation A.3 *In S_{OPT} the second order must be of type b.*

Proof: Let $(k+1) > 2$ be the smallest position of a type-b order in S_{OPT}. Following the convention, let b_1 denote this type-b order. It will be shown that there is a schedule S which has b_1 in a position less than $(k+1)$ but larger than 1, such that

$$\sum_{j=1}^{3n} C_j(S) < \sum_{j=1}^{3n} C_j(S_{OPT}).$$

In S_{OPT} the first order must be of type a, and the remaining $(k-1)$ orders are either of type a or type c. Let the number of type-a and type-c orders be n_a and n_b, respectively. Clearly, $n_a + n_c = k$.

If $n_c = 0$, it means that all of the first k orders are of type a. Thus, if $k = 2$, the schedule for the first $(k+1)$ orders is $a_1a_2b_1$. Consider the subsequence of S_{OPT} before c_1. This subsequence is denoted by:

$$S_{OPT} : a_1a_2b_1(a/b)^lc_1,$$

where $l \geq 0$. It can be shown that if S_{OPT} is changed to S:

$$S : a_1b_1c_1(a/b)^la_2,$$

then the $\sum C_j$ is lower. Thus, it may be assumed that $k \geq 3$. For $k \geq 3$, the schedule for the first $(k+1)$ orders is:

$$S_{OPT} : a_1(a)^{k-2}a_kb_1.$$

It can be shown that

$$S : a_1(a)^{k-2}b_1a_k$$

is better than S_{OPT}. Thus, it may be assumed that $n_c \geq 1$.

It may also be assumed that $n_a \geq 2$. Indeed, if $n_a = 1$, the schedule for the first $(k+1)$ order is:

$$S_{OPT} : a_1c_1(c)^{k-2}b_1.$$

It can be shown that

$$S : a_1b_1c_1(c)^{k-2}.$$

has smaller $\sum C_j$ than S_{OPT}. Thus, it may be assumed that $k = n_a + n_c \geq 3$.

For $k = 3$, there are only two possible schedules for the first $(k+1)$ orders: $a_1a_2c_1b_1$ and $a_1c_1a_2b_1$. In the former case $(a_1a_2c_1b_1)$, it can be shown that the schedule $a_1b_1c_1a_2$ is a better schedule. In the latter case $(a_1c_1a_2b_1)$, it can be shown that the schedule $a_1c_1a_2b_1$ is a better schedule.

Thus, from now on, it needs to focus only on the case $k \geq 4$.

First of all, for the first $(k+1)$ orders,

$$F^{(1)}_{k+1} = \left(2n_aL + \sum_{j=1}^{n_a} a_j\right) + n_cL + (2L + 2X + b_1). \tag{A.16}$$

$$F^{(2)}_{k+1} = \left(2n_aL + n_aX - \sum_{j=1}^{n_a} a_j\right) + \left(2n_cL + 2\sum_{j=1}^{n_c} c_j\right) + (L + X - b_1). \tag{A.17}$$

$$F^{(3)}_{k+1} = n_c(5L + 2X) + \sum_{j=1}^{n_c} c_j. \tag{A.18}$$

<u>Case 1</u>: $F^{(1)}_{k+1} \geq F^{(2)}_{k+1}$ and $F^{(1)}_{k+1} \geq F^{(3)}_{k+1}$.

From (A.16) and (A.17),

$$F^{(1)}_{k+1} \geq F^{(2)}_{k+1} \Rightarrow (n_c = 0) \textbf{ or } (n_c = 1 \textbf{ and } n_a \leq 0). \tag{A.19}$$

Since it is assumed that $n_c \geq 1$ and $k \geq 4$, this leads to a contradiction.

<u>Case 2</u>: $F^{(2)}_{k+1} \geq F^{(1)}_{k+1}$ and $F^{(2)}_{k+1} \geq F^{(3)}_{k+1}$.

From (A.16) and (A.17),

$$F^{(1)}_{k+1} \leq F^{(2)}_{k+1} \Rightarrow (n_c = 1 \textbf{ and } n_a \geq 3) \textbf{ or } (n_c \geq 2). \tag{A.20}$$

It should be noted that, if $n_c = 1$, then $n_a \geq 3$, since it needs to focus only on $k = n_a + n_c \geq 4$.

From (A.17) and (A.18),

$$F_{k+1}^{(3)} \leq F_{k+1}^{(2)} \Rightarrow n_c \leq \frac{2n_a + 1}{3}. \tag{A.21}$$

Look at the order in the k^{th} position, it must have:

$$F_k^{(1)} = \left(2n_a L + \sum_{j=1}^{n_a} a_j\right) + n_c L. \tag{A.22}$$

$$F_k^{(2)} = \left(2n_a L + n_a X - \sum_{j=1}^{n_a} a_j\right) + \left(2n_c L + 2\sum_{j=1}^{n_c} c_j\right). \tag{A.23}$$

Clearly, under the condition of (A.20), $F_k^{(1)} < F_k^{(2)}$ always holds. There are the following two cases:

Case 2(a): The order in the k^{th} position is of type a.

In this case, the first $(k+1)$ orders of the schedule is:

$$a_1(a/c)^{k-2} a_{n_a} b_1.$$

It can be shown that:

$$a_1(a/c)^{k-2} b_1 a_{n_a}.$$

is a better schedule.

Case 2(b): The order in the k^{th} position is of type c.

In this case, it can be proved by contradiction that the order in the $(k-1)^{st}$ position must be of type a. Suppose that a_{n_a} is in the $(k-l)^{th}$ $(2 \leq l \leq n_c)$ position, then the first k orders in S_{OPT} is:

$$S_{OPT} : a_1(a/c)^{k-l-2} a_{n_a} c_{n_c-l+1} c^{l-2} c_{n_c}.$$

It can be shown that S:

$$S : a_1(a/c)^{k-l-2} c_{n_c-l+1} a_{n_a} c^{l-2} c_{n_c}$$

is better than S_{OPT}.

Thus, the $(k-1)^{st}$ order must be of type a if the k^{th} order is of type c. It follows that the first $(k+1)$ orders are scheduled as:

$$S_{OPT}: a_1(a/c)^{k-3}a_{n_a}c_{n_c}b_1.$$

It can be shown that

$$S: a_1(a/c)^{k-3}b_1c_{n_c}a_{n_a}$$

is better than S_{OPT}

<u>Case 3</u>: $F^{(3)}_{k+1} \geq F^{(1)}_{k+1}$ and $F^{(3)}_{k+1} \geq F^{(2)}_{k+1}$.

For this case, the first $(k+1)$ orders in S_{OPT} must be of the pattern:

$$S_{OPT}: a_1(a/c)^{l}c_{n_c}(a)^{k-l-2}b_1.$$

It can be shown that the following pattern S:

$$S: a_1(a/c)^{l}(a)^{k-l-2}b_1c_{n_c}.$$

has a smaller $\sum C_j$ than S_{OPT}.

Summarizing the above cases, the correctness of the observation follows. $\square$

Observation A.4 *In S_{OPT} the third order must be of type c.*

Proof: Let $(k+1) > 3$ be the smallest position of a type-c order in S_{OPT}. Following the convention, let c_1 denote this type-c order. It will be shown that there is a schedule S which has c_1 in a position less than $(k+1)$ but larger than 2, such that

$$\sum_{j=1}^{3n} C_j(S) < \sum_{j=1}^{3n} C_j(S_{OPT}).$$

It is clear that the first k orders are either of type a or type b. Let the number of type-a and type-b orders be n_a and n_b, respectively. Clearly,

$$(k+1) > 3 \Rightarrow k = n_a + n_b \geq 3.$$

Therefore, c_1 must be finished on either machine 1 or machine 2.

For $k = 3$, there are two possible patterns for S_{OPT}: $a_1b_1a_2c_1$ and $a_1b_1b_2c_1$. In the former case, it can be shown that $a_1b_1c_1a_2$ is better than $a_1b_1a_2c_1$. In the latter case, it can be shown that $a_1b_1c_1b_2$ is better than $a_1b_1b_2c_1$.

From now on, it can be focused on $k \geq 4$ only. There are two cases to consider:

Case 1: $n_a \geq 2$.

The pattern of the first $(k+1)$ orders in S_{OPT} is

$$S_{OPT} : a_1b_1(a/b)^{k-l-3}a_{a_n}b^lc_1.$$

It can be shown that S:

$$S : a_1b_1(a/b)^{k-l-3}c_1b^la_{a_n}.$$

is better than S_{OPT}.

Case 2: $n_a < 2$.

$n_a < 2$ also means that $n_a = 1$, since the first order must be a type-a order. Thus,

$$k = n_a + n_b \geq 4 \Rightarrow n_b \geq 3.$$

The pattern of the first $(k+1)$ orders in S_{OPT} is

$$S_{OPT} : a_1b_1(b)^{k-3}b_{b_n}c_1.$$

It can be shown that S:

$$S : a_1b_1(b)^{k-3}c_1b_{n_b}$$

is better than S_{OPT}.

Summarizing the above cases, the correctness of the observation follows. $\square$

Now it is ready to show that the orders are scheduled in S_{OPT} in the order (abc), repeated n times. This can be shown by induction on k, where k is the number of times that the (abc) pattern repeats. The previous observations have shown that the lemma is true for $k = 1$. The inductive step can be shown by the same argument as in the previous observations. This follows from the observations that (1) At the beginning, all three machines start at time 0; (2) After scheduling the (abc) pattern k times, the finish times of the three machines are $5L + 2X + \alpha_1$, $5L + 2X + \alpha_2$, and $5L + 2X + \alpha_3$, respectively, where α_i $(1 \leq i \leq 3)$ is a linear combination of the a_i's, b_i's and c_i's. Since L and X are much larger than the a_i's, b_i's and c_i's, one can view the finish times of the three machines as more or less equal. In other words, the a_i's, b_i's and c_i's are inconsequential in the arguments.

APPENDIX B

PROOF OF LEMMA 5.6

This appendix presents the proof of Lemma 5.6 in Chapter 5.

Without loss of generality, it is assumed for order j that

$$p_{1j} \geq p_{2j} \geq \ldots \geq p_{kj}.$$

For the convenience of description, the following notation is defined:

- Let $W_1^{(i)}(l, j)$ denote the i^{th} smallest workload before product type l of order j is assigned by the *LPT* rule based on profile A.
- Let $W_2^{(i)}(l, j)$ denote the i^{th} smallest workload before product type l of order j is assigned by the *LPT* rule based on profile B.
- Let $C_{lj}(A)$ denote the finish time of product type l of order j assigned by the *LPT* rule based on profile A.
- Let $C_{lj}(B)$ denote the finish time of product type l of order j assigned by the *LPT* rule based on profile B.

It is clear that the *LPT* rule does not introduce idle time on the machines as long as there are product types available for producing. Thus, if the *LPT* rule assigns the product types of order j based on profile A and profile B, respectively, it will be shown that, for each product type $l = 1, 2, \ldots, k$ of order j, it satisfies:

$$W_1^{(i)}(l, j) \leq W_2^{(i)}(l, j), i = 1, 2, \ldots, m; \tag{B.1}$$

and

$$C_{lj}(A) \leq C_{lj}(B). \tag{B.2}$$

The proof is by induction. First consider the base case where $l = 1$. By the assumption that for each $i = 1, 2, \ldots m$ the i^{th} smallest workload of profile A is not larger than that of profile B, it is easy to see that both (B.1) and (B.2) are satisfied. As the inductive hypothesis, it is assumed that both (B.1) and (B.2) hold for each product type $l = 1, 2, \ldots, x$. Now look at product type $x + 1$. Since (B.1) holds before product type x is assigned based on the two profiles, respectively, the two smallest workloads of the two profiles satisfy:

$$W_1^{(1)}(x,j) \leq W_2^{(1)}(x,j), \quad i = 1, 2, \ldots, m. \tag{B.3}$$

Now after x is assigned to the machines with the smallest workload in the two profiles, respectively, let

$$W_1 = W_1^{(1)}(x,j) + p_{xj}, \text{ and } W_2 = W_2^{(1)}(x,j) + p_{xj}. \tag{B.4}$$

It is clear that (B.3) and (B.4) imply

$$W_1 \leq W_2. \tag{B.5}$$

As long as x is assigned, it needs to determine $W_1^{(i)}(x + 1, j)$ and $W_2^{(i)}(x + 1, j)$, $i = 1, 2, \ldots, m$. Clearly, there must exist some i_1 $(1 \leq i_1 \leq m)$ and i_2 $(1 \leq i_2 \leq m)$ such that

$$W_1^{(i_1)}(x,j) \leq W_1 \leq W_1^{(i_1+1)}(x,j), \text{ and } W_2^{(i_2)}(x,j) \leq W_2 \leq W_2^{(i_2+1)}(x,j).$$

Three cases are considered.

Case 1: $i_1 = i_2$. $W_1^{(i)}(x + 1, j)$ and $W_2^{(i)}(x + 1, j)$ are determined as follows:

(a) For each $i = 1, 2, \ldots, i_1 - 1$ let

$$W_1^{(i)}(x + 1, j) = W_1^{(i+1)}(x,j), \text{ and } W_2^{(i)}(x + 1, j) = W_2^{(i+1)}(x,j).$$

(b) For $i = i_1 = i_2$, let

$$W_1^{(i_1)}(x + 1, j) = W_1, \text{ and } W_2^{(i_1)}(x + 1, j) = W_2.$$

(c) For each $i = i_1 + 1, \ldots, m$, let

$$W_1^{(i)}(x+1,j) = W_1^{(i)}(x,j), \text{ and } W_2^{(i)}(x+1,j) = W_2^{(i)}(x,j).$$

After x has been assigned to the two profiles, it is clear that

$$W_1^{(1)}(x+1,j) \le W_1^{(2)}(x+1,j) \le \ldots \le W_1^{(m)}(x+1,j),$$

and

$$W_2^{(1)}(x+1,j) \le W_2^{(2)}(x+1,j) \le \ldots \le W_2^{(m)}(x+1,j).$$

From the above assignment, it is clear that (B.1) is still satisfied before $x+1$ is assigned.

Case 2: $i_1 > i_2$. $W_1^{(i)}(x+1,j)$ and $W_2^{(i)}(x+1,j)$ can be determined as follows:

(a) For each $i = 1, 2, \ldots, i_2 - 1$, let

$$W_1^{(i)}(x+1,j) = W_1^{(i+1)}(x,j), \text{ and } W_2^{(i)}(x+1,j) = W_2^{(i+1)}(x,j). \tag{B.6}$$

(b) Let

$$W_1^{(i_2)}(x+1,j) = W_1^{(i_2+1)}(x,j), \text{ and } W_2^{(i_2)}(x+1,j) = W_2. \tag{B.7}$$

(c) For each $i = i_2 + 1, \ldots, i_1 - 1$, let

$$W_1^{(i)}(x+1,j) = W_1^{(i+1)}(x,j), \text{ and } W_2^{(i)}(x+1,j) = W_2^{(i)}(x,j). \tag{B.8}$$

(d) Let

$$W_1^{(i_1)}(x+1,j) = W_1, \text{ and } W_2^{(i_1)}(x+1,j) = W_2^{(i_1)}(x,j). \tag{B.9}$$

(e) Finally, for each $i = i_1 + 1, \ldots, m$, let

$$W_1^{(i)}(x+1,j) = W_1^{(i)}(x,j), \text{ and } W_2^{(i)}(x+1,j) = W_2^{(i)}(x,j). \tag{B.10}$$

Clearly, according to (B.6), for each $i = 1, 2, \ldots, i_2 - 1$

$$W_1^{(i)}(x+1, j) \le W_2^{(i)}(x+1, j).$$

For $i = i_2, \ldots, i_1$, from (B.7), (B.8), and (B.9)

$$\begin{aligned} W_1^{(i_2)}(x+1,j) &\le W_1^{(i_2+1)}(x+1,j) \le \ldots \le W_1^{(i_1-1)}(x+1,j) \le W_1^{(i_1)}(x+1,j) = W_1 \\ &\le W_2 = W_2^{(i_2)}(x+1,j) \le W_2^{(i_2+1)}(x+1,j) \le \ldots \le W_2^{(i_1)}(x+1,j). \end{aligned}$$

It certainly implies that

$$W_1^{(i)}(x+1,j) \le W_2^{(i)}(x+1,j) \text{ for each } i = i_2, \ldots, i_1.$$

For each $i = i_1 + 1, \ldots, m$, (B.10) also implies that

$$W_1^{(i)}(x+1, j) \le W_2^{(i)}(x+1, j).$$

Thus, in this case, (B.1) still holds before $x + 1$ is assigned.

Case 3: $i_1 < i_2$. $W_1^{(i)}(x+1,j)$ and $W_2^{(i)}(x+1,j)$ are determined as follows:

(a) For each $i = 1, 2, \ldots, i_1 - 1$, let

$$W_1^{(i)}(x+1,j) = W_1^{(i+1)}(x,j), \text{ and } W_2^{(i)}(x+1,j) = W_2^{(i+1)}(x,j). \tag{B.11}$$

(b) Let

$$W_1^{(i_1)}(x+1,j) = W_1, \text{ and } W_2^{(i_1)}(x+1,j) = W_2^{(i_1+1)}(x,j). \tag{B.12}$$

(c) For each $i = i_1 + 1, \ldots, i_2 - 1$, let

$$W_1^{(i)}(x+1,j) = W_1^{(i)}(x,j), \text{ and } W_2^{(i)}(x+1,j) = W_2^{(i+1)}(x,j). \tag{B.13}$$

(d) Let

$$W_1^{(i_2)}(x+1,j) = W_1^{(i_2)}(x,j), \text{ and } W_2^{(i_2)}(x+1,j) = W_2. \tag{B.14}$$

(e) Finally, for each $i = i_2 + 1, \dots, m$, let

$$W_1^{(i)}(x+1, j) = W_1^{(i)}(x, j), \text{ and } W_2^{(i)}(x+1, j) = W_2^{(i)}(x, j). \tag{B.15}$$

According to (B.11), for each $i = 1, 2, \dots, i_1 - 1$ it is easy to see

$$W_1^{(i)}(x+1, j) \leq W_2^{(i)}(x+1, j).$$

When $i = i_1$,

$$W_1^{(i_1)}(x+1, j) = W_1 \leq W_1^{(i_1+1)}(x, j) \leq W_2^{(i_1+1)}(x, j) = W_2^{(i_1)}(x+1, j).$$

For each $i = i_1 + 1, \dots, i_2 - 1$,

$$W_1^{(i)}(x+1, j) = W_1^{(i)}(x, j) \leq W_2^{(i)}(x, j) \leq W_2^{(i+1)}(x, j) = W_2^{(i)}(x+1, j).$$

When $i = i_2$,

$$W_1^{(i_2)}(x+1, j) = W_1^{(i_2)}(x, j) \leq W_2 = W_2^{(i_2)}(x+1, j).$$

Finally, for each $i = i_2 + 1, \dots, m$, it is easy to see

$$W_1^{(i)}(x+1, j) \leq W_2^{(i)}(x+1, j).$$

Thus, in this case, (B.1) also holds before $x + 1$ is assigned.

It can be concluded from the above three cases that (B.1) holds before $x+1$ is assigned. After product type $x + 1$ is assigned based on the two profiles, respectively, it is easy to see that

$$C_{x+l,j}(A) = W_1^{(1)}(x+1, j) + p_{x+1,j} \leq W_2^{(1)}(x+1, j) + p_{x+1,j} = C_{x+1,j}(B).$$

Thus, (B.2) also holds for $l = x + 1$.

The above inductive proof shows that (B.2) holds for each $l = 1, 2, \ldots, k$. It follows that

$$C_j(A) = \max_{1 \leq l \leq k}\{C_{lj}(A)\} \leq \max_{1 \leq l \leq k}\{C_{lj}(B)\} = C_j(B).$$

This completes the proof.

REFERENCES

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, and S. Khanna. Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates. In *The Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pp. 32–43, 1999.

[2] S. Arora and C. Lund. Hardness of Approximations. In D.S. Hochbaum (ed.), *Approximation Algorithms for NP-Hard Problems*, pp. 399–446. PWS Publishing Company, 1996.

[3] J. Barnes, M. Laguna, and F. Glover. An Overview of Tabu Search Approaches to Production Scheduling Problems. In D. Brown and W. Scherer (eds.), *Intelligent Scheduling Systems*, pp. 101–127. Kluwer Academic Publishers, 1995.

[4] J. Blocher and D. Chhajed. The Customer Order Lead-Time Problem on Parallel Machines. *Naval Research Logistics*, vol. 43, pp. 629–654, 1996.

[5] P. Brucker. *Scheduling Algorithms.* Springer Verlag, Berlin, First Edition, 1995.

[6] P. Brucker and S. Knust. Complexity Results for Scheduling Problems. http://www.mathematik.uni-osnabrueck.de/research/OR/class/.

[7] J. Bruno, E. G. Coffman, and R. Sethi. Scheduling Independent Tasks to Reduce Mean Finishing Time. *Communications of the ACM*, vol. 17, pp. 382–387, 1974.

[8] C. Chekuri and S. Khanna. Approximation Algorithms for Minimizing Average Weighted Completion Time. In J. Y-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press, Boca Raton, FL, USA, 2004.

[9] T. C. E. Cheng, C. T. Ng, and J. J. Yuan. A Stronger Complexity Result for the Single Machine Multi-Operation Jobs Scheduling Problem to Minimize the Number of Tardy Jobs. *Journal of Scheduling*, vol. 6, pp. 551–555, 2003.

[10] T. C. E. Cheng and G. Wang. Customer Order Scheduling on Multiple Facilities. Technical Report 11/98-9, Faculty of Business and Information Systems, The Hong Kong Polytechnic University, Hong Kong, 1999.

[11] E. G. Coffman, M. R. Garey, and D. S. Johnson. An Application of Bin-Packing to Multiprocessor Scheduling. *SIAM Journal on Computing*, vol. 7, pp. 1–17, 1978.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, USA, Second Edition, 2001.

[13] G. Dobson. Scheduling Independent Tasks on Uniform Processors. *SIAM Journal on Computing*, vol. 13, pp. 705–716, 1984.

[14] J. Du and J. Y-T. Leung. Minimizing Total Tardiness on One Machine is NP-Hard. *Mathematics of Operations Research*, vol. 15, pp. 483–495, 1990.

[15] I. Duenyas. Estimating the Throughput of a Cyclic Assembly System. *International Journal of Production Research*, vol. 32, pp. 403–1410, 1994.

[16] H. Emmons. One-Machine Sequencing to Minimize Certain Functions of Job Tardiness. *Operations Research*, vol. 17, pp. 701–715, 1969.

[17] A. Fréville. The Multidimensional 0-1 Knapsack Problem: An Overview. *European Journal of Operational Research*, vol. 155, pp. 1–21, 2004.

[18] D. K. Friesen. Tighter Bounds for LPT Scheduling on Uniform Processors. *SIAM Journal on Computing*, vol. 16, pp. 554-560, 1987.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[20] A. E. Gerodimos, C. N. Potts, and T. Tautenhahn. Scheduling Multi-Operation Jobs on a Single Machine. *Annals of Operations Research*, vol. 92, pp. 87–105, 1999.

[21] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, vol. 1, pp. 190–206, 1989.

[22] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing*, vol. 2, pp. 4–32, 1990.

[23] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Boston, USA, 1997.

[24] T. Gonzalez, O. H. Ibarra, and S. Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, vol. 6, pp. 155–166, 1977.

[25] R. L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, vol. 45, pp. 1563–1581, 1966.

[26] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416–429, 1969.

[27] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[28] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms. *Mathematics of Operations Research*, vol. 22, pp. 513–544, 1997.

[29] D. S. Hochbaum. Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems. In D. S. Hochbaum (ed.), *Approximation Algorithms for NP-Hard Problems*, pp. 94–143. PWS Publishing Company, 1996.

[30] D. S. Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1996.

[31] E. C. Horvath, S. Lam, and R. Sethi. A Level Algorithm for Preemptive Scheduling. *Journal of the Association for Computing Machinary*, vol. 24, pp. 32–43, 1977.

[32] J. R. Jackson. Scheduling a Production Line to Minimize Maximum Tardiness. Technical Report 43, Management Science Research Project, University of California, Los Angeles, 1955.

[33] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, vol. 9, pp. 256–278, 1974.

[34] F. M. Julien and M. J. Magazine. Scheduling Customer Orders – An Alternative Production Scheduling Approach. *Journal of Manufacturing and Operations Management*, vol. 3, pp. 177–199, 1990.

[35] H. Kise, T. Ibaraki, and H. Mine. A Solvable Case of the One-Machine Scheduling Problem with Ready and Due Times. *Operations Research*, vol. 26, pp. 121–126, 1978.

[36] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, vol. 13, pp. 32–44, 1992.

[37] M. Laguna. *Implementing and Testing the Tabu Cycle and Conditional Probability Methods*, 2004. Working Paper.

[38] E. L. Lawler. Optimal Sequencing of a Single Machine Subject to Precedence Constraints. *Management Science*, vol. 19, pp. 544–546, 1973.

[39] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. In S.C. Graves, A.H.G Rinnooy Kan, and P.H. Zipkin (eds.), *Handbooks in Operations Research and Management Science*, vol. 4, pp. 445–522. Elsevier Science Publisher, 1993.

[40] C. Y. Lee, T. C. E. Cheng, and B. M. T. Lin. Minimizing the Makespan in the 3-Machine Assembly-Type Flowshop Scheduling Problem. *Management Science*, vol. 39, pp. 616–625, 1993.

[41] J. K. Lenstra. Sequencing by Enumerative Methods. Technical Report 69, Mathematical Centre Tracts, Amsterdam, Netherland, 1977.

[42] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of Machine Scheduling Problems. *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[43] J. Y-T. Leung (ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* Chapman and Hall/CRC Press, Boca Raton, FL, USA, 2004.

[44] J. Y-T. Leung, C. Y. Lee, C. W. Ng, and G. H. Young. Minimizing Total Weighted Completion Time on Flexibile Machines for Order Scheduling. Submitted.

[45] J. Y-T. Leung, H. Li, and M. L. Pinedo. Order Scheduling in a Flexible Environment with Parallel Resources and no Setup Times. Submitted.

[46] J. Y-T. Leung, H. Li, and M. L. Pinedo. Order Scheduling in an Environment with Dedicated Resources in Parallel. *Journal of Scheduling*. Accepted for publication.

[47] J. Y-T. Leung, H. Li, and M. L. Pinedo. Scheduling Orders for Multiple Product Types to Minimize Total Weighted Completion Time. Submitted.

[48] J. Y-T. Leung, H. Li, and M. L. Pinedo. Scheduling Orders for Multiple Product Types with Due Date Related Objectives. *European Journal of Operational Research*. Accepted for publication.

[49] J. Y-T. Leung, H. Li, M. L. Pinedo, and C. Sriskandarajah. Open Shops with Jobs Overlap – Revisited. *European Journal of Operational Research*, vol. 163, pp. 569–571, 2005.

[50] J. Y-T. Leung, H. Li, M. L. Pinedo, and J. Zhang. Minimizing Total Weighted Completion Time when Scheduling Orders in a Flexible Environment with Uniform Machines. Submitted.

[51] J. M. Moore. An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. *Management Science*, vol. 15, pp. 102–109, 1968.

[52] R. R. Muntz and E. G. Coffman. Optimal Preemptive Scheduling on Two-Processor Systems. *IEEE Transactions on Computers*, vol. 11, pp. 1014–1020, 1969.

[53] R. R. Muntz and E. G. Coffman. Preemptive Scheduling of Real Time Tasks on Multiprocessor Systems. *Journal of the Association for Computing Machinary*, vol. 17, pp. 324–338, 1970.

[54] C. T. Ng, T. C. E. Cheng, and J. J. Yuan. Strong NP-hardness of the Single Machine Multi-Operation Jobs Total Completion Time Scheduling Problem. *Information Processing Letters*, vol. 82, pp. 187–191, 2002.

[55] C. T. Ng, T. C. E. Cheng, and J. J. Yuan. Concurrent Open Shop Scheduling to Minimize the Weighted Number of Tardy Jobs. *Journal of Scheduling*, vol. 6, pp. 405–412, 2003.

[56] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.

[57] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Second Edition, 2002.

[58] C. N. Potts, S. V. Sevast'janov, V. A. Strusevich, L. N. Wassenhove, and C. M. Zwaneveld. The Two-Stage Assembly Scheduling Problem: Complexity and Approximation. *Operations Research*, vol. 43, pp. 346–355, 1995.

[59] M. Queranne. Structure of a Simple Scheduling Polyhedron. *Mathematical Programming*, vol. 58, pp. 263–285, 1993.

[60] S. Rajagopalan and V. V. Vazirani. Primal-Dual RNC Approximation Algorithms for Set Cover and Covering Integer Programs. *SIAM Journal on Computing*, vol. 28, pp. 525–540, 1998.

[61] S. Sahni. *Data Structures, Algorithms, and Applications in C++*. WCB/McGraw-Hill, 1998.

[62] W. E. Smith. Various Optimizers for Single Stage Production. *Naval Research Logistics Quarterly*, vol. 3, pp. 59–66, 1956.

[63] C. S. Sung and S. H. Yoon. Minimizing Total Weighted Completion Time at a Pre-Assembly Stage Composed of Two Feeding Machines. *International Journal of Production Economics*, vol. 54, pp. 247–255, 1998.

[64] V. V. Vazirani. *Approximation Algorithms*. Springer, New York, 2003.

[65] E. Wagneur and C. Sriskandarajah. Open Shops with Jobs Overlap. *European Journal of Operational Research*, vol. 71, pp. 366–378, 1993.

[66] G. Wang and T. C. E. Cheng. Customer Order Scheduling to Minimize Total Weighted Completion Time. In *Proceedings of the 1st Multidisciplinary Conference on Scheduling Theory and Applications*, pp. 409–416, 2003.

[67] J. Yang. *Scheduling with Batch Objectives*. PhD thesis, Industrial and Systems Engineering Graduate Program, The Ohio State University, Columbus, Ohio, 1998.

[68] J. Yang and M. E. Posner. Scheduling Parallel Machines for the Customer Order Problem. *Journal of Scheduling*, vol. 8, pp. 49–74, 2005.

[69] M. Zweben and M. Fox, editors. *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.