Algorithmic And Mathematical Programming Approaches To Scheduling Problems With Energy-Based Objectives

Kan Fang
Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/open_access_dissertations
Part of the Industrial Engineering Commons, and the Operational Research Commons

Recommended Citation
Fang, Kan, "Algorithmic And Mathematical Programming Approaches To Scheduling Problems With Energy-Based Objectives" (2013). Open Access Dissertations. 132.
https://docs.lib.purdue.edu/open_access_dissertations/132

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
This is to certify that the thesis/dissertation prepared
By Kan Fang

Entitled
ALGORITHMIC AND MATHEMATICAL PROGRAMMING APPROACHES TO SCHEDULING
PROBLEMS WITH ENERGY-BASED OBJECTIVES

For the degree of Doctor of Philosophy

Is approved by the final examining committee:
Andrew L. Liu
Yanjun Li
Fu Zhao
Yuehwern Yih

To the best of my knowledge and as understood by the student in the Research Integrity and
Copyright Disclaimer (Graduate School Form 20), this thesis/dissertation adheres to the provisions of
Purdue University’s “Policy on Integrity in Research” and the use of copyrighted material.

Approved by Major Professor(s): Andrew L. Liu

Approved by: Steven Landry 11/21/2013
Head of the Graduate Program Date
ALGORITHMIC AND MATHEMATICAL PROGRAMMING APPROACHES TO
SCHEDULING PROBLEMS WITH ENERGY-BASED OBJECTIVES

A Dissertation
Submitted to the Faculty
of
Purdue University
by
Kan Fang

In Partial Fulfillment of the
Requirements for the Degree
of
Doctor of Philosophy

December 2013
Purdue University
West Lafayette, Indiana
ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my major advisor, Dr. Nelson A. Uhan, for his excellent guidance and support during my PhD study. His strong enthusiasm for research and his earnest mentorship is of great value to me. I am grateful for his comments, inspiration, and patience throughout these years, all of which led me to a deeper understanding of the topics in this dissertation.

I would also like to thank my co-advisor, Dr. Andrew L. Liu, for his guidance and his time to help me a lot during the last two years of my PhD study. I am thankful to Dr. Fu Zhao and Dr. John W. Sutherland for their collaboration on our work and inspiration on my research topics in this dissertation. I also thank Dr. Yanjun Li and Dr. Yuehwern Yih for graciously giving their time and sharing their knowledge in support of completing this dissertation. I wish to thank Dr. Thomas L. Morin, Dr. Hong Wan and Dr. Mihailo Stojnic for being a teaching assistant of their courses. I also appreciate the help of Cheryl Barnhart and Steven Duket making my study at Purdue smoothly.

Finally, I would like to thank my parents, my brother, my good friends Fang Yi and Dr. Cheng Ouyang for their love and encouragement. Thanks to Jikai Zou, Zhiying Gu, Xiao Zhang and Hao Zhang for their help and discussions.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vii</td>
</tr>
</tbody>
</table>

## I Overview

1. Introduction ........................................... 1
   1.1 Outline of the dissertation ..................... 5
2. Background ............................................. 8
   2.1 Flow shop scheduling ............................. 8
   2.2 Scheduling problems with energy, power and electricity cost criteria 10
3. Mathematical descriptions of the problems .... 17
   3.1 Permutation flow shop scheduling with peak power consumption constraints ........................................... 17
   3.2 Scheduling on a single machine under time-of-use tariffs 19

## II Permutation flow shop scheduling with peak power consumption constraints

4. Discrete speeds and unlimited intermediate storage: integer programming formulations .......... 22
   4.1 Disjunctive formulation ............................ 24
   4.2 Assignment and positional formulation .......... 26
     4.2.1 Lower and upper bounds for start and completion time decision variables .................. 27
     4.2.2 Basic AP formulation ........................... 28
     4.2.3 Strengthening the basic AP formulation: concurrent job valid inequalities .................. 29
     4.2.4 Strengthening the basic AP formulation: nondelay valid inequalities ....................... 32
   4.3 Experimental study ..................................... 35
     4.3.1 Computational environment ....................... 35
4.3.2 Two heuristic algorithms for finding feasible schedules  ....  37
4.3.3 Experiment 1: makespan ($C_{\text{max}}$) vs. power consumption ($Q_{\text{max}}$)  ..  38
4.3.4 Experiment 2: disjunctive vs. basic AP formulation  ....  40
4.3.5 Experiment 3: basic AP vs. enhanced AP formulation  ..  41
4.3.6 Tables  ............................................  43

5 Two machines, and zero intermediate storage  .................  50
5.1 Discrete speeds  ........................................  50
5.2 Continuous speeds  ......................................  52
5.2.1 Arbitrary work volume of jobs across machines  ..........  52
5.2.2 Consistent work volume of jobs across machines  ......  54
5.2.3 Equal work volume of jobs across machines  .............  57

III Scheduling on a single machine under time-of-use tariffs

6 The uniform-speed machine case  ......................................  60
6.1 NP-hardness and inapproximability of Problem SMSEC-U ....  60
6.2 Properties of optimal schedules for Problem SMSEC-U-pmtn  ..  62
6.3 An exact polynomial-time algorithm for Problem SMSEC-U-pyr 64

7 The continuous speed-scalable machine case  .....................  70
7.1 Properties of optimal schedules for Problem SMSEC-S-pmtn  ..  70
7.2 NP-hardness of Problem SMSEC-S  ..........................  75
7.3 Inapproximability of Problem SMSEC-S  .........................  78
7.4 Structural properties of optimal schedules for Problem SMSEC-S with $|\mathcal{P}| = 2$  .....................  84
7.4.1 Case 1: single job  ....................................  84
7.4.2 Case 2: two jobs with equal work volume  ............  86
7.5 Approximation algorithms for Problem SMSEC-S ..............  89
7.6 Experimental study  .........................................  95
7.6.1 Experiment 1: ratio of electricity cost vs. ratio of electricity prices  ..................  96
7.6.2 Experiment 2: performance evaluation of Algorithms 7.5.1 and 7.5.2  ..........  98
7.6.3 Tables  ............................................  100

IV Conclusion

8 Concluding remarks and future directions  .......................  103

LIST OF REFERENCES  ........................................  105

VITA  ........................................  111
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Comparison of average running time between disjunctive and basic AP formulations.</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Speedup factor between disjunctive and basic AP formulations.</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Initial sizes of the basic and the enhanced AP formulations.</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Results for instances with intermediate values of $Q_{\text{max}}$ and $m = 2, n = 8, d = 2$.</td>
<td>44</td>
</tr>
<tr>
<td>4.5</td>
<td>Results for solved instances with low or high $Q_{\text{max}}$.</td>
<td>45</td>
</tr>
<tr>
<td>4.6</td>
<td>Results for solved instances with intermediate $Q_{\text{max}}$.</td>
<td>46</td>
</tr>
<tr>
<td>4.7</td>
<td>Average optimality gap for instances with $m = 2, d = 2$.</td>
<td>47</td>
</tr>
<tr>
<td>4.8</td>
<td>Average optimality gap for instances with $m = 2, d = 5$.</td>
<td>48</td>
</tr>
<tr>
<td>4.9</td>
<td>Average optimality gap for instances with $m = 4, d = 2$.</td>
<td>49</td>
</tr>
<tr>
<td>6.1</td>
<td>An instance with pyramidal TOU tariffs.</td>
<td>65</td>
</tr>
<tr>
<td>7.1</td>
<td>Average ratio between electricity costs obtained by Algorithm 7.5.1, 7.5.2, 7.5.3 and the corresponding optimal preemptive schedule.</td>
<td>100</td>
</tr>
<tr>
<td>7.2</td>
<td>Average ratio between electricity costs obtained by Algorithm 7.5.1 and the optimal preemptive schedule.</td>
<td>101</td>
</tr>
<tr>
<td>7.3</td>
<td>Average ratio between electricity costs obtained by Algorithm 7.5.2 and the optimal preemptive schedule.</td>
<td>101</td>
</tr>
<tr>
<td>7.4</td>
<td>Average ratio between electricity costs obtained by Algorithm 7.5.2 and Algorithm 7.5.1.</td>
<td>102</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 A variable TOU tariff scheme (Braithwait et al. 2007)</td>
<td>2</td>
</tr>
<tr>
<td>1.2 A representative supply/dispatch stack for a US region with 65 GW of dispatchable generating capacity (Chase 2012)</td>
<td>3</td>
</tr>
<tr>
<td>3.1 An example of the TOU tariffs</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Gantt chart for Example 4.1</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Cast iron plates with slots</td>
<td>35</td>
</tr>
<tr>
<td>4.3 Trade-off between makespan and peak power consumption</td>
<td>39</td>
</tr>
<tr>
<td>4.4 Relationship between running time and peak power consumption</td>
<td>39</td>
</tr>
<tr>
<td>4.5 Average speedup factors between the disjunctive and basic AP formulations</td>
<td>41</td>
</tr>
<tr>
<td>6.1 The TOU tariff used in the proof of Theorem 6.1</td>
<td>61</td>
</tr>
<tr>
<td>6.2 A pairwise interchange on two units of jobs $i$ and $j$</td>
<td>63</td>
</tr>
<tr>
<td>6.3 An example of pyramidal TOU tariffs</td>
<td>64</td>
</tr>
<tr>
<td>6.4 An optimal schedule for the instance given in Example 6.1</td>
<td>65</td>
</tr>
<tr>
<td>6.5 A schedule generated by Algorithm 6.3.1</td>
<td>67</td>
</tr>
<tr>
<td>7.1 Jobs in period $l$ in an optimal preemptive schedule</td>
<td>71</td>
</tr>
<tr>
<td>7.2 Job $j$ is processed in both periods in an optimal schedule</td>
<td>78</td>
</tr>
<tr>
<td>7.3 Job $j$ (the grey part) processed in an optimal preemptive schedule $\sigma_{\text{pmtn}}$</td>
<td>90</td>
</tr>
<tr>
<td>7.4 Job $j$ (the grey part) processed in a preemptive schedule $\sigma_{\text{pmtn}}^m$</td>
<td>93</td>
</tr>
<tr>
<td>7.5 Electricity costs obtained by Algorithms 7.5.1, 7.5.2 and 7.5.3 for an instance</td>
<td>98</td>
</tr>
<tr>
<td>7.6 Average ratio between electricity costs obtained by Algorithms 7.5.1, 7.5.2 and their corresponding optimal preemptive schedule when we fix $n = 5$</td>
<td>100</td>
</tr>
<tr>
<td>7.7 Average ratio between electricity costs obtained by Algorithms 7.5.1, 7.5.2 and their corresponding optimal preemptive schedule when we fix $K = 50$</td>
<td>100</td>
</tr>
</tbody>
</table>
ABSTRACT

Fang, Kan Ph.D., Purdue University, December 2013. Algorithmic and mathematical programming approaches to scheduling problems with energy-based objectives. Major Professors: Nelson A. Uhan and Andrew L. Liu.

This dissertation studies scheduling as a means to address the increasing concerns related to energy consumption and electricity cost in manufacturing enterprises. Two classes of problems are considered in this dissertation: (i) minimizing the makespan in a permutation flow shop with peak power consumption constraints (the PFSPP problem for short) and (ii) minimizing the total electricity cost on a single machine under time-of-use tariffs (the SMSEC problem for short). We incorporate the technology of dynamic speed scaling and the variable pricing of electricity into these scheduling problems to improve energy efficiency in manufacturing. The challenge in the PFSPP problem is to keep track of which jobs are running concurrently at any time so that the peak power consumption can be properly taken into account. The challenge in the SMSEC problem is to keep track of the electricity prices at which the jobs are processed so that the total electricity cost can be properly computed.

For the PFSPP problem, we consider both mathematical programming and combinatorial approaches. For the case of discrete speeds and unlimited intermediate storage, we propose two mixed integer programs and test their computational performance on instances arising from the manufacturing of cast iron plates. We also examine the PFSPP problem with two machines and zero intermediate storage, and investigate the structural properties of optimal schedules in this setting.

For the SMSEC problem, we consider both uniform-speed and speed-scalable machine environments. For the uniform-speed case, we prove that this problem is strongly NP-hard, and in fact inapproximable within a constant factor, unless $P = NP$. 
In addition, we propose an exact polynomial-time algorithm for this problem when all the jobs have the same work volume and the electricity prices follow a so-called pyramidal structure. For the speed-scalable case, in which jobs can be processed at an arbitrary speed with a trade-off between speed and energy consumption, we show that this problem is strongly NP-hard and that there is no polynomial time approximation scheme for this problem. We also present different approximation algorithms for this case and test the computational performance of these approximation algorithms on randomly generated instances.
Part I
Overview
1. INTRODUCTION

In the United States, about one-third of all the end-use energy consumption is associated with industrial activities. As a result, improving energy-saving manufacturing technology is crucial as our world faces increasing energy costs and energy security challenges. It is well-known that electricity is an efficient and safe way to move energy from one place to another, and most countries use it as the main energy source for manufacturing (Park et al. 2009). So, both electricity-intensive customers and providers have huge opportunities to save costs by improving the efficiency of electricity consumption. Recently, two approaches have been proposed to create energy efficient systems: exploiting the variable pricing of electricity and implementing the technology of dynamic speed scaling.

Due to the increasing deployment of advanced metering and monitoring infrastructures, the smart grid system has been widely implemented. Under this system, during peak demand when power generation approaches its limit, the supplier can contact consumers to alert them of an outage or power interruption. In order to improve the reliability and efficiency of electrical power grids, more and more electricity suppliers have begun to implement variable pricing to manage the balance between electricity supply and demand. For example, in time-of-use (TOU) tariffs, retail energy prices to customers vary hourly to reflect changes in wholesale energy prices, which are typically announced a day ahead or an hour ahead. Such price structures are used to shift electricity use from peak hours to off-peak hours. Figure 1.1 shows a typical variable TOU tariff scheme.

In a typical TOU tariff scheme, as the demand increases, the cost goes up disproportionately as the suppliers have to include older, more inefficient generation plants and more expensive and nonrenewable resources to meet the demand (Shapiro and
Figure 1.1.: A variable TOU tariff scheme (Braithwait et al. 2007).

Tomain 2005). As a result, the variation in electricity prices can be as high as a factor of 10 from one hour to the next. Figure 1.2 illustrates the marginal cost of power from each source, ordered by their priority in the supply stack. The wide differences in prices between peak and off-peak hours encourage customers to reduce or reschedule their demand in response to the electricity prices and avoid consuming electricity during hours of peak prices. Reacting to such electricity price fluctuations can lead to high cost savings, in particular for large energy consumers such as data centers or manufacturing production lines.

On the other hand, retail electricity rates have traditionally been set at the same level for broad classes of customers, and at a fixed flat price that reflects the broad average of the hourly costs to serve customers in the class over a year or a season. For example, electricity suppliers also offer single flat pricing or seasonal flat pricing schemes for customers who prefer price certainty and are unwilling to bear price risk. Such static, averaged retail rates suggest that it is also important under flat rate schemes to consider the objectives of minimizing energy and power consumption. Until recently, most efforts aimed at minimizing energy and power consumption in manufacturing have been focused on developing machines and equipment that are more energy and power efficient (e.g. Dornfeld and Wright 2007; Haapala et al. 2009;
Figure 1.2.: A representative supply/dispacth stack for a US region with 65 GW of dispatchable generating capacity (Chase 2012).

Nava et al. 2010; Diarra et al. 2010). However, several researchers have observed that in various manufacturing enterprises, the energy and power consumption by a machine for the active removal of material can be quite small compared to the background process needed for operating the machine (e.g. Dahmus and Gutowski 2004; Kordonowy 2002). Drake et al. (2006) showed that whenever a machine or a component is turned on, there is a significant amount of start up energy consumption, and confirmed that when a machine is idle a significant amount of energy is consumed. Gutowski et al. (2005) showed that in a mass production environment, more than 85% of the energy is used for functions that are not directly related to the actual production of parts. All of this implies that significant energy savings can be found by finding alternate operational strategies, i.e., the smarter management and scheduling of tasks, rather than focusing on updating individual machines or processes to be more energy efficient. Finding and implementing alternate operational strategies has the additional benefit of requiring fairly small capital investments. Unfortunately,
current shop scheduling strategies adopted by manufacturing enterprises focus mainly on productivity and time; in almost all cases, energy and environmental factors are not considered.

Although energy and power consumption have rarely been considered as a major factor in manufacturing systems, minimizing energy and power consumption has been an area of interest in mobile and large-scale computing systems since the mid-1990s. Most of this research assumes that reducing the average power consumption proportionately decreases energy costs. However, peak power consumption – that is, the maximum power consumption over all time instants – also plays a key role in the energy costs of electricity-intensive manufacturing enterprises. For example, it is common for electricity contracts to include a charge for the customer’s peak power over a billing period, in addition to the energy consumption (Chase 2012). As a result, manufacturing enterprises may reduce their costs tremendously by practicing peak power management.

Research on using scheduling to reduce energy and power consumption and electricity costs in manufacturing is rather sparse. In this dissertation, we will explore the use of scheduling as a means to reduce the energy and power consumption as well as electricity costs of manufacturing enterprises.

First, we formulate models and design algorithms that find optimal or near-optimal schedules for shop scheduling problems with energy and power related objectives. Inspired by the nature of energy costs in manufacturing environments, we consider several variants of the multi-objective problem of minimizing the makespan and the peak power consumption in a permutation flow shop, in which jobs must be processed in the same order on every machine in the shop, with machines that can be run at varying speeds, and hence can be run to consume varying amounts of power. We consider the problem when speeds are discrete and when they are continuous. In addition, we consider flow shops with both zero and unlimited intermediate storage between machines. (In a flow shop with zero intermediate storage, a completed job cannot leave the current machine until the next machine is available.) For
simplicity, we refer to this problem in general as the *permutation flow shop scheduling problem with peak power consumption constraints*, or the PFSPP problem for short.

Note that optimizing energy consumption and optimizing electricity costs can be quite different under variable electricity pricing schemes like TOU tariffs. In this dissertation, we also explore scheduling jobs on a single machine to minimize the total electricity cost under TOU tariffs. We consider the problem where jobs can only be processed at a uniform speed and where jobs can be processed at an arbitrary speed with a trade-off between speed and energy consumption. We refer to these machine types as *uniform-speed* and *speed-scalable* respectively. For simplicity, we refer to this problem in general as the *single machine scheduling problem with electricity costs*, or the SMSEC problem for short.

1.1 Outline of the dissertation

In Chapter 2 we review the literature on flow shop scheduling and scheduling with objectives related to energy and power consumption and electricity costs. We pay special attention to previous work that uses dynamic speed scaling techniques to reduce energy and power consumption and electricity costs. In Chapter 3, we formally define the different scheduling problems that we study in this dissertation.

We consider both mathematical programming and combinatorial approaches to the PFSPP problem in Chapters 4 and 5. Unlike most classical scheduling problems, we need to be able to keep track of which jobs are running concurrently at any time in order to take the peak power consumption into account. This presents some interesting modeling and algorithmic challenges, and some of our results may be of interest in other scheduling applications (e.g. Thörnblad 2013).

For the case of discrete speeds and unlimited intermediate storage, we propose two integer programming models in Chapter 4, inspired by existing formulations for shop scheduling problems (Manne 1960; Wagner 1959; Lasserre and Queyranne 1992). In order to strengthen these formulations, we give valid inequalities that exploit the
structure of optimal schedules and the properties of concurrently running jobs. We also test the computational performance of these two formulations and the effectiveness of these valid inequalities on a set of instances based on the manufacture of cast iron plates with slots.

We examine the PFSPP problem with two machines and zero intermediate storage in Chapter 5. When speeds are discrete, we show that this problem is equivalent to a special case of the asymmetric traveling salesperson problem. In addition, we also consider the PFSPP problem with two machines, zero intermediate storage, and continuous speeds. We make the common assumption that power consumption is an exponential function of speed. We show that this problem also can be transformed to an equivalent asymmetric traveling salesperson problem. Moreover, if the jobs have some special features, we obtain combinatorial polynomial time algorithms for finding optimal schedules.

In Chapter 6, we consider the SMSEC problem with uniform-speed machine. We show that this problem is strongly NP-hard. In addition, we show that this problem is inapproximable within a constant factor, unless P = NP. However, when all the jobs have the same work volume and the TOU tariffs are so-called pyramidal—that is, the electricity price monotonically increases until it reaches its highest value and then monotonically decreases—we can obtain an exact polynomial-time algorithm for this problem.

In Chapter 7, we consider the SMSEC problem with speed-scalable machine, in which jobs can be processed at an arbitrary speed with a trade-off between speed and energy consumption. We first consider the preemptive version of this problem, in which we use the structural properties of an optimal schedule to formulate the problem as a convex program and apply the Karush-Kuhn-Tucker (KKT) optimality conditions to obtain an optimal preemptive schedule. We then show that this problem is strongly NP-hard by exploiting the structure of an optimal preemptive schedule. In addition,
we show that there is no polynomial time approximation scheme (PTAS)\(^1\) for this problem. Finally, we present and analyze different approximation algorithms\(^2\) that find non-preemptive schedules by transforming preemptive schedules into non-preemptive ones, and empirically test the performance of these approximation algorithms on a set of randomly generated instances.

This dissertation is partly based on the following prior work. Fang et al. (2011a) and Fang et al. (2011b) proposed the mixed integer programs for the PFSSPP problem studied in Chapter 4. Fang et al. (2013a) proposed valid inequalities for these formulations, studied their computational performance, also presented here in Chapter 4, and investigated the PFSSPP problem with two machines and zero intermediate storage, presented here in Chapter 5. Fang et al. (2013b) considered the SMSEC problem studied here, investigated the computational complexity of the SMSEC problem with uniform-speed machine and gave exact polynomial-time algorithms under some special cases as presented here in Chapter 6. In addition, Fang et al. (2013b) studied the computational complexity of the SMSEC problem with speed-scalable machine and proposed and analyzed approximation algorithms for these problems as presented here in Chapter 7.

---

\(^1\)A polynomial time approximation scheme (PTAS) is an algorithm that for any \(\varepsilon > 0\), finds a solution whose objective value is within a factor \((1 + \varepsilon)\) of the optimal value, and whose running time is polynomial in the input size.

\(^2\)A \(\rho\)-approximation algorithm is an algorithm that finds a solution whose objective value is within a factor \(\rho\) of the optimal value, and whose running time is polynomial in the input size. The factor \(\rho\) is known as the performance guarantee.
2. BACKGROUND

To the best of our knowledge, this dissertation is one of the first in-depth studies of (i) flow shop scheduling with criteria related to both time and power and (ii) single machine scheduling with criteria related to electricity cost under time-of-use tariffs. Since the scheduling literature is quite vast, we focus our review in this chapter on the existing research on flow shop scheduling and scheduling with energy, power and electricity cost related criteria.

2.1 Flow shop scheduling

In a flow shop, jobs have to be processed on a set of machines in the same route, i.e. each job has to be processed first on machine 1, then on machine 2, and so on. In a flow shop, each job may be characterized by different parameters, including processing times on each of the machines, a release date, a deadline or a due date. Because of their extensive applications in various industrial fields, flow shop scheduling problems have been investigated by an enormous number of researchers, starting with the work by Johnson (1954). Researchers have developed various algorithms and techniques to solve different flow shop problems with diverse machine characteristics and different objective functions. In Chapter 1 we described the permutation flow shop environment and constraints on intermediate storage between machines. Here we introduce some other common features of flow shops that have been studied.

In a flow shop, each machine may either be a classical machine that can process only one job at a time, or a batching machine that can process several jobs simultaneously. For a batching machine, the completion time of the entire batch is determined by the job with the longest processing time. Potts (2000) gave a comprehensive review of the literature on scheduling with batching.
In some manufacturing shops, the setup time of a job may not be independent of the job already being processed; that is, the setup time may depend on both the job to be processed and the immediately preceding job. In this case, we cannot simply consider the setup time as part of the processing time. Research has shown that the setup times sometimes are significant for the effective management of the manufacturing process. For more information on flow shop scheduling problems with setup considerations, we refer the reader to the review by Allahverdi et al. (2008).

Another phenomenon that may occur in a flow shop is the no-wait requirement. In this case, jobs are not allowed to wait between two successive machines, which may be important in some manufacturing enterprises. For example, in a steel rolling mill, a slab of steel is not allowed to wait between machines since it would cool off during a wait, rendering the steel unprocessable by the next machine. For example, Naderi et al. (2012) studied multi-objective no-wait flow shop scheduling problems that minimize both the makespan and the total tardiness of the jobs.

Aside from traditional flow shop problems, manufacturers often encounter more complex machine environments, which can be viewed as generalizations of the flow shop. One such generalization is the flexible flow shop, also called the hybrid flow shop, which has multiple stages in the shop: each job has to be processed first at stage 1, then at stage 2, and so on. At each stage there is a number of identical machines in parallel, and each job can be processed on any of the machines at that stage. For more details on flexible flow shops, we refer the reader to a survey by Linn and Zhang (1999). Another type of generalization of the flow shop is called the job shop, which does not require the jobs to be processed on m machines in the same route; instead, in a job shop, each job has its own predetermined route of machines to follow. Jain and Meeran (1999) provided a comprehensive survey on deterministic job shop scheduling problems, including an overview of the history, the techniques used and the researchers involved over the last 40 years.

In order to solve shop scheduling problems, many different techniques have been proposed and investigated, which can be mainly categorized into two types, exact
methods and approximate methods. Exact methods, such as mathematical programming, branch-and-bound, and dynamic programming, have been successfully applied to tackle small-sized flow shop problems. For example, Pan (1997) presented a study of different integer programming formulations for flow shop and permutation flow shop scheduling problems. Moursli and Pochet (2000) proposed a branch-and-bound algorithm to solve a flexible flow shop scheduling problem. However, medium- and large-sized flow shop problems are often too hard to solve exactly with today's technology. For these problems, approximate methods, such as tabu search, simulated annealing and genetic algorithms have been applied to obtain good-quality schedules in a reasonable amount of time. For example, Grabowski and Pempera (2005) developed some local search algorithms such as tabu search and descending search algorithms for no-wait flow shop scheduling problems with the makespan objective. Suresh and Mohanasundaram (2006) proposed a metaheuristic procedure based on simulated annealing to find Pareto-optimal solutions to permutation flow shop scheduling problems with the objectives of minimizing makespan and total flow time.

For more details on the extensive work done on flow shop scheduling problems, we refer the reader to the following surveys. Hejazi and Saghafian (2005) gave a comprehensive survey of flow shop scheduling problems with the makespan objective. Gupta and Stafford (2006) presented the evolution of flow shop scheduling problems and possible approaches for solving these problems. Sun et al. (2011a) gave a comprehensive survey of state-of-the-art approaches for multi-objective flow shop scheduling problems.

2.2 Scheduling problems with energy, power and electricity cost criteria

Recall that energy consumption is the integral of power consumption over time, and that electricity cost is the integral of consumed electrical energy times the (potentially time-varying) electricity price per unit over time. Much research has focused on scheduling jobs in order to decrease average power consumption, or equivalently, total energy consumption. However, relatively little work has been done on scheduling
to decrease peak power consumption and electricity cost under time-of-use tariffs. We will review two branches of the literature on scheduling with energy, power and electricity cost criteria that arose from (1) mobile and large scale computing systems and (2) manufacturing systems.

There is a considerable body of literature on scheduling computer processors in a way that minimizes total energy and average power consumption. Two techniques called *dynamic speed scaling* (also called *dynamic voltage scaling*) and *power-down* are currently widely used in practice. In dynamic speed scaling, the processors can run at varying speeds: reducing the speed of a processor lowers power consumption, but results in longer processing time. Therefore, with dynamic speed scaling, the scheduling problem is to decide not only which jobs should be processed, but also which speed to use. In power-down, there usually exist thresholds that specify the length of idle time after which a system is powered down. In the following, we will concentrate on the dynamic speed scaling literature, since these techniques are closely related to the problems that will be studied in this dissertation. We refer the reader to the surveys by Benini et al. (2000) and Irani and Pruhs (2005) for details on power-down techniques. For more pointers to the literature on other power saving techniques (e.g. clock gating, asynchronous logic), we refer the reader to the surveys by Brooks et al. (2000) and Mudge (2001).

In most of the research on dynamic speed scaling, it is assumed that the processor speed can be chosen arbitrarily and the associated power consumption is an exponential function of the speed, which is typical for CMOS devices. Yao et al. (1995) initiated the algorithmic study of formulating speed scaling problems as scheduling problems. They investigated the problem of minimizing energy consumption subject to hard job deadlines on a single processor. This variant of the speed scaling problem with deadline feasibility has been studied extensively in the literature. For example, Albers et al. (2007) investigated various multi-processor variants of this problem. They considered both offline and online scenarios, and proposed and analyzed several different algorithms. Chen et al. (2011) considered the problem of unrelated parallel
machine scheduling using dynamic speed scaling techniques with a given energy budget. They presented approximation algorithms to minimize the makespan, assuming each job has a release time on each machine. Angel et al. (2012) studied the problem of scheduling jobs with release dates and deadlines on parallel speed-scalable machines to minimize the total energy consumption. They considered the cases where jobs can be processed preemptively and where jobs can be processed with migration (i.e. a job can be interrupted and allowed to continue its execution on a different machine), and proposed polynomial time algorithms for these problems. Greiner et al. (2009) investigated the problem of scheduling jobs on multiple speed-scalable machines without migration to minimize the total energy consumption. They proposed the first constant factor online and offline approximation algorithms for the problem in which jobs have arbitrary release times and deadlines. Situations without deadline requirements but with other objectives and constraints have also been studied in this literature. For example, Pruhs et al. (2008) considered the problem of minimizing the makespan on parallel machines with a fixed energy budget, where there are precedence constraints between tasks. In addition, other power functions have also been considered in the literature (e.g. Bansal et al. 2009).

Researchers have put a particular focus on different variants of scheduling problems with energy, power and electricity cost related criteria in a single machine environments. Albers and Fujiwara (2007) studied the problem of scheduling jobs on a speed-scalable machine to minimize the power consumption plus the total flow time of all the jobs. Bansal et al. (2007a) studied the design and performance of speed scaling algorithms for scheduling jobs with deadlines on a single machine to address concerns with energy and temperature. Bansal et al. (2007b) designed and analyzed online algorithms for the problem of minimizing weighted flow time plus energy consumption on a single machine with preemptive jobs. Bampis et al. (2012) considered the problem of scheduling jobs nonpreemptively on a single machine to minimize the maximum lateness plus energy consumption and proved that different variants of the problem in the presence of arbitrary release dates are strongly NP-hard. Antoniadis and Huang
(2013) showed that the non-preemptive version of the single-machine problem of Yao et al. (1995) is strongly NP-hard, and provided an approximation algorithm with a constant performance guarantee for this problem. Bampis et al. (2013) explored the idea of transforming a preemptive schedule into a non-preemptive one and proposed an approximation algorithm for the problem of scheduling jobs on a single speed scaling machine to minimize total energy consumption. Kulkarni and Munagala (2013) investigated scheduling jobs online and preemptively to minimize the weighted flow time plus electricity cost on a single machine, and showed that these problems are significantly different from their counterparts without electricity costs.

Aside from the above models in which jobs can be processed at an arbitrary continuous speeds, two other speed-scaling models have also been studied in the literature, namely the \textit{bounded speed model} in which the speeds are within a given bounded interval, and the \textit{discrete speed model} in which the speeds of jobs can only be chosen within a set of discrete speeds. For example, Chan et al. (2007) considered online algorithms for energy-efficient deadline scheduling on a single machine with a bounded speed set. Li and Yao (2005) gave an exact algorithm for the same problem of Yao et al. (1995) when there are only discrete speeds available. Chen et al. (2005) showed that minimizing energy consumption with a discrete speed set while meeting all deadlines is NP-hard. Kwon and Kim (2005) proposed a voltage allocation technique for discrete supply voltages to produce a preemptive task schedule that minimizes total energy consumption. Bunde (2009) investigated the problem of minimizing makespan on a single machine with a discrete speed set, and proposed an algorithm that finds all Pareto optimal schedules, given different energy budgets. For more details on the speed scaling literature with other machine environments and objectives, we refer the reader to the work by Irani and Pruhs (2005), Albers (2010), and Bampis et al. (2013).

Peak power consumption has also received some attention in computer science community, since it affects the power supply and cooling technologies in the design of computer processors. However, unlike the literature discussed above, little work has been done on using scheduling as a means of minimizing peak power consumption.
Some researchers proposed techniques similar to dynamic speed scaling and tested them in different environments arising in computing systems. Felter et al. (2005) presented power shifting techniques to reduce the peak power consumption of servers, which dynamically distributes power among components using workload-sensitive policies. Isci et al. (2006) evaluated the performance of different policies for global dynamic power management under various objectives with a global power consumption budget, and showed that these policies perform significantly better than static management, in which the peak power consumption budget is viewed as a soft limitation that can temporarily be exceeded. Sartori and Kumar (2009) presented a gradient ascent-based technique to decrease peak power consumption for multi-core architectures. They showed that their proposed techniques show up to 47% improvements in throughput for a given power consumption budget. Kontorinis et al. (2009) proposed a new architecture that uses adaptive processors to decrease peak power consumption by a table-driven approach. They show that this architecture can cut peak power consumption by 25% while maintaining voltage variation below 5%. Note that all of the work described above do not formally formulate their problems as minimizing the peak power consumption in a scheduling problem.

One type of scheduling problems that is related to the SMSEC problem studied in this dissertation is the time slot scheduling problem, which originally arose from settings in mobile telecommunication systems and wireless sensor networks. These problems seek schedules that optimize energy consumption and other quality of service objectives. In these problems, the time horizon is generally divided into time slots of equal duration, each with a given maximum capacity, and each job may only be assigned to a given subset of the slots. For example, Kannan and Wei (2006) studied the minimum energy consumption scheduling problem for duty-cycle and rate-constrained wireless sensor node transmission over equal duration time slots. Detti et al. (2009) proposed a strongly polynomial time algorithm for the problem of maximizing the number of scheduled jobs in time slots. Unlike the above time slot scheduling problems with job availability constraints, the SMSEC problem requires
us to keep track of the electricity prices at which jobs are processed, since in our setting the cost for processing a job in different time slots can be quite different. Most related to the SMSEC problem, Wan and Qi (2010) considered a related problem of scheduling jobs on a single machine with unit length time slots to minimize a linear combination of total time slot costs and a traditional scheduling objective, in which each time slot has a corresponding cost.

In contrast to the scheduling literature in the computer science community, research on using scheduling to reduce energy consumption in manufacturing is rather sparse. Mouzon et al. (2007) considered a CNC machine in a shop making small aircraft parts and investigated scheduling jobs on this machine in order to minimize its total energy consumption. They observed that changing the normal practice, i.e. leaving the non-bottleneck machines idle, can lead to up to an 80% savings on total idle, start up, and shut down energy consumption. They also proposed a multi-objective mathematical programming model to minimize the energy consumption and total completion time on a single CNC machine. In follow-up work, Mouzon and Yildirim (2008) proposed a metaheuristic algorithm to minimize the total energy consumption and total tardiness on a single machine. There does not appear to be much work on shop scheduling problems with objectives related to energy consumption. One exception we found is the work by Subaï et al. (2006), who considered energy consumption and waste generation in hoist scheduling problems arising from surface treatment processes.

Our literature review also suggests that little work has been done on using scheduling to reduce power consumption in manufacturing, except for some work related to steel-making plants. One of the earliest works is by Boukas et al. (1991), who proposed a solution method for a nonstandard scheduling problem arising from a steel-making plant with a global power consumption constraint on the group of machines. The machines process jobs in batch mode and require a given amount of energy at each fusion phase in a production cycle. They formulated the scheduling problem as a combined optimal control mathematical programming problem, and proposed a two-level hierarchical algorithm to solve it. Zhang and Tang (2010) studied scheduling
as a means to minimize the total cost of power consumption with power supply capacity constraints on identical parallel machines. The paper presents a subgradient method-based Lagrangian relaxation algorithm, and shows that the proposed methods can generate near-optimal schedules with average duality gaps less than 7%. Recently, an increasing amount of work has considered using the technology of dynamic speed scaling or multi-power states on energy-aware schedules in manufacturing enterprises (e.g. Sun et al. 2011b; Fang et al. 2011b, 2013a).

As we mentioned before, optimizing energy consumption and optimizing electricity costs can be quite different under TOU tariffs. Research on using scheduling to reduce electricity costs in manufacturing is also very sparse. One exception is the work by Sharma et al. (2013), who considered the electricity cost and the environmental impact of a multi-part multi-machine scheduling problem under a time-of-use tariff.

Note that in the speed scaling literature, it is typically assumed that each job needs to be processed on a single processor or one of multiple parallel processors. This is to be expected, as this matches typical computing environments. However, in a typical manufacturing environment, jobs often need to be processed on multiple machines in some order; in other words, in some kind of job shop environment. As a result, much of the work on speed scaling is not directly applicable to the problems faced by manufacturing enterprises. To the best of our knowledge, it appears that no one has studied flow shop scheduling problems with peak power consumption criteria, especially with dynamic speed scaling. In addition, as we mentioned above, there has been little work done on scheduling jobs on a single machine to minimize electricity cost. In this dissertation, we aim to begin to fill these gaps. In particular, in the subsequent chapters, we study the algorithmic aspects (i.e. computational complexity, mathematical programming approaches, combinatorial algorithms) of flow shop scheduling problems with objectives based on both time and power (the PFSPPP problem) and single machine scheduling problems with electricity cost objectives under time-of-use tariffs (the SMSEC problem).
3. MATHEMATICAL DESCRIPTIONS OF THE PROBLEMS

3.1 Permutation flow shop scheduling with peak power consumption constraints

As we mentioned in the introduction, we refer to the permutation flow shop scheduling problem that we study in this dissertation as the permutation flow shop scheduling problem with peak power consumption constraints (or the PFSPP problem for short). An instance of the PFSPP problem consists of a set $\mathcal{J} = \{1, 2, \ldots, n\}$ of jobs and a set $\mathcal{M} = \{1, 2, \ldots, m\}$ of machines. Each job $j$ on machine $i$ has required work volume $p_{ij}$, and must be processed nonpreemptively first on machine 1, then on machine 2, and so on, and jobs must be processed in the same order on every machine. There is a set of speeds $\mathcal{S}$: a job $j \in \mathcal{J}$ processed on machine $i \in \mathcal{M}$ at speed $s \in \mathcal{S}$ has an associated processing time $p_{ij}$ and power consumption $q_{ij}$. In addition, we are given a threshold $Q_{\text{max}}$ on the total power consumption at any time instant of the schedule. We make the following assumptions on the problem input.

Assumption 3.1 We assume that when we process a job at a higher speed, its processing time decreases, while its power consumption increases. That is, as $s$ increases, $p_{ij}$ decreases and $q_{ij}$ increases. In addition, we assume that the power consumption associated with processing a job on a machine at a particular speed is constant from the job’s start time until but not including the job’s completion time. Finally, we assume that $\min_{s \in \mathcal{S}} \{q_{ij}\} \leq Q_{\text{max}}$ for all $i \in \mathcal{M}$ and $j \in \mathcal{J}$.

We define a feasible schedule as a schedule in which the total power consumption at any time instant is no more than the given threshold $Q_{\text{max}}$. In this dissertation, we focus on finding a feasible schedule minimizes the makespan $C_{\text{max}}$, the completion
time of the last job on the last machine $m$, for the PFSPPP problems. Depending on
the type of speed set and flow shop environment, we define the following variants of
the PFSPPP problem.

**Problem PFSPPP-DU** The set of speeds $\mathcal{S} = \{s_1, s_2, \ldots, s_d\}$ is discrete. The flow
shop has unlimited intermediate storage. The relationship between processing time,
power consumption, and speed can be arbitrary as long as it satisfies Assumption 3.1.
Find a feasible schedule that minimizes the makespan $C_{\text{max}}$.

Without loss of generality, we assume that $s_1 < s_2 < \cdots < s_d$. As we know, finding
a schedule with minimum makespan for Problem PFSPPP-DU is computationally
difficult: when $m = 3$, the problem of simply minimizing makespan in a permutation
flow shop is already NP-hard (Garey and Johnson 1979).

Note that in manufacturing systems, a flow shop may have limited intermediate
storage (i.e. a buffer) between two successive machines; that is, when the intermediate
storage is full, a machine may not be allowed to release a completed job until the next
machine is available. In this dissertation, we consider a variant of Problem PFSPPP-DU,
in which the flow shop has two machines and zero intermediate storage.

**Problem PFSPPP-DTZ** The set of speeds $\mathcal{S} = \{s_1, s_2, \ldots, s_d\}$ is discrete. The flow
shop has two machines, that is, $\mathcal{M} = \{1, 2\}$, and zero intermediate storage. The
relationship between processing time, power consumption, and speed can be arbitrary
as long as it satisfies Assumption 3.1. Find a feasible solution that minimizes the
makespan $C_{\text{max}}$.

Unlike in Problems PFSPPP-DU and PFSPPP-DTZ, it might be the case that each
job can be processed at an arbitrary speed within a given continuous range. It is
typical to have power as an exponential function of speed (e.g. Brooks et al. 2000;
Mudge 2001; Bouzid 2005). We also consider the following two machine variant of the
PFSPPP problem with zero intermediate storage.
Problem PFSPP-CTZ The set of speeds $\mathcal{S} = [s_{\text{min}}, s_{\text{max}}]$ is continuous. The flow shop has two machines, that is, $\mathcal{M} = \{1, 2\}$, and zero intermediate storage. Each job $j$ processed on machine $i$ at speed $s \in \mathcal{S}$ has processing time $p_{ij} = p_{ij}/s$ and power consumption $q_{ij} = s^\alpha$ for some constant $\alpha > 1$. Find a feasible schedule that minimizes the makespan $C_{\text{max}}$.

We will discuss our research on Problem PFSPP-DU in Chapter 3, and our research on Problems PFSPP-DTZ and PFSPP-CTZ in Chapter 5.

3.2 Scheduling on a single machine under time-of-use tariffs

As we mentioned in the introduction, we refer to the electricity cost scheduling problem that we study in this dissertation as the single machine scheduling problem with electricity costs (or the SMSEC problem for short). In this problem, there is a time-of-use (TOU) tariff scheme that consists of a set $\mathcal{P} = \{1, 2, \ldots, K\}$ of time periods, with each period $k \in \mathcal{P}$ having a per-unit electricity price of $c_k$ and a duration of $d_k$. Figure 3.1 shows a general TOU tariff scheme. For simplicity’s sake, we define $T = \sum_{k=1}^{K} d_k$ as the length of the time horizon. In addition, there is a set $\mathcal{J} = \{1, 2, \ldots, n\}$ of jobs. Each job $j$ has required work volume $w_j$ and required power consumption $q_j$. We assume that the work volume of jobs and the durations of time periods are given as integers.

Figure 3.1.: An example of the TOU tariffs.
In this dissertation, we focus on minimizing the total electricity cost $E$, which is calculated on the basis of consumed electrical energy over time, taking into account that each time period has a corresponding electricity price per unit energy consumed. For instance, a job $j$ that is processed for $t$ time units in period $k$ incurs an electricity cost of $c_k q_j t$. We define a feasible schedule as a schedule in which all of the jobs are processed within the time horizon. Depending on the machine type and job environment, we define the following variants of the SMSEC problems.

**Problem SMSEC-U** Jobs must be processed non-preemptively at a uniform speed; that is, once a job has started, it must be processed until its completion. Each job $j \in J$ has processing time $p_j = w_j$ and power consumption $q_j$. The relationship between processing time and power consumption is arbitrary. Given a TOU tariff scheme, find a feasible schedule that minimizes the total electricity cost $E$.

**Problem SMSEC-U-pmtn** Jobs can be processed preemptively at a uniform speed; that is, a job can be paused and resumed at a later point of time during a schedule. Each job $j \in J$ has processing time $p_j = w_j$ and power consumption $q_j$. The relationship between processing time and power consumption is arbitrary. Given a TOU tariff scheme, find a feasible schedule that minimizes the total electricity cost $E$.

We also consider a variant of Problem SMSEC-U, in which all the jobs have the same work volume and the electricity prices in the given TOU tariff scheme satisfy $c_1 < c_2 < \cdots < c_{h-1} < c_h > c_{h+1} > \cdots > c_K$. We call such tariffs *pyramidal* TOU tariffs.

**Problem SMSEC-U-pyr** Jobs must be processed non-preemptively at a uniform speed. Each job $j \in J$ has processing time $p_j = p$ and power consumption $q_j$. The relationship between processing time and power consumption is arbitrary. Given a pyramidal TOU tariff scheme, find a feasible schedule that minimizes the total electricity cost $E$. 
Unlike the uniform-speed cases in Problems SMSEC-U, SMSEC-U-pmtn, and SMSEC-U-pyr, in which we only consider exploiting the variable pricing of electricity, we will also consider the following variants of the SMSEC problem that incorporate the technology of dynamic speed scaling. As we know, in a dynamic speed scaling setting, the power consumption for processing a job depends on the speed at which the machine runs: the lower the speed, the higher the power consumption. For simplicity of analysis with little loss of applicability, it is typical to assume that power is an exponential function of speed (e.g. Brooks et al. 2000; Mudge 2001; Bouzid 2005).

**Problem SMSEC-S-pmtn** Jobs can be processed preemptively at an arbitrary speed. Specifically, when a job is processed across several time periods in a preemptive schedule, the speeds for processing different parts of the job can be different. When one part of job \( j \) is processed in period \( l \) at speed \( s \) with work volume \( w_{jl} \), its processing time in period \( l \) is \( p_{jl} = w_{jl}/s \), and its power consumption in period \( l \) is \( q_j = s^{\alpha} \) for some constant \( \alpha > 1 \). Given a TOU tariff scheme, find a feasible schedule that minimizes the total electricity cost \( E \).

**Problem SMSEC-S** Jobs must be processed non-preemptively at an arbitrary speed. Each job \( j \in J \) has work volume \( w_j \). When job \( j \) is processed at speed \( s \), its processing time is \( p_j = w_j/s \), and its power consumption is \( q_j = s^{\alpha} \) for some constant \( \alpha > 1 \). Given a TOU tariff scheme, find a feasible schedule that minimizes the total electricity cost \( E \).

We will discuss our research on Problems SMSEC-U, SMSEC-U-pmtn and SMSEC-U-pyr in Chapter 6, and our research on Problems SMSEC-S-pmtn and SMSEC-S in Chapter 7.
Part II
Permutation flow shop scheduling with peak power consumption constraints
4. DISCRETE SPEEDS AND UNLIMITED INTERMEDIATE STORAGE: INTEGER PROGRAMMING FORMULATIONS

A great deal of research has focused on solving flow shop scheduling problems with traditional time-based objectives using integer programming approaches. These efforts have been primarily based on two families of mixed-integer linear programs (MILPs). One is due to Manne’s (1960) formulation, that uses linear ordering variables and pairs of dichotomous constraints (called the disjunctive constraints) to ensure one job is processed before another or vice versa. The other is based on Wagner’s (1959) developments in the use of classical assignment problem to assign jobs to positions on machines. Various researchers have investigated the computational performance of different mixed integer programs for the permutation flow shop scheduling problem based on these two families with respect to several traditional time-based objectives (e.g. Stafford et al. 2005; Keha et al. 2009; Unlu and Mason 2010).

In this chapter, we consider the multi-objective problem of minimizing the makespan and the peak power consumption in a permutation flow shop. We search for Pareto optimal schedules, or schedules for which no other schedule has both lower makespan and lower peak power consumption. In order to handle the bicriteria nature of this problem, we fix an upper bound on the peak power consumption, and minimize the makespan of the schedule. We propose two integer programming models for Problem PFSPP-DU, i.e., the PFSPP problem with discrete speeds and unlimited intermediate storage, which are based on the work of Manne (1960) and Wagner (1959). We compare the performance of these two integer programming models and discover some promising formulation paradigms that can subsequently be applied to solve larger scheduling problems under power consumption constraints.
Unlike most ordinary flow shop scheduling problems, we need to keep track of jobs that are running concurrently on machines at any time instant. For this reason we cannot apply the existing integer programming models for the ordinary permutation flow shop problem directly. Note that in Problem PFSPP-DU, each job must be processed nonpreemptively with exactly one speed $s \in S$ on each machine. As a result, when a job is started on a given machine, the power consumption of that machine will stay the same until this job is finished. For any time instance $t$, let $J_t$ be the set of jobs being processed at $t$. Let $C^L_t (S^L_t)$ be the completion (start) time of the last job completed (started) before time $t$. Let $C^R_t (S^R_t)$ be the completion (start) time of the first job completed (started) after time $t$. Denote $t_1 = \max\{C^L_t, S^L_t\}$ and $t_2 = \min\{C^R_t, S^R_t\}$. Then for any time $t'$ within $[t_1, t_2)$, we have $J_{t'} = J_t$; that is, the total power consumption in the flow shop is a constant between $t_1$ and $t_2$.

**Example 4.1** Suppose at time $t$, there are three jobs $J_t = \{j, k, l\}$ that are processed concurrently (see Figure 4.1). $S^L_t$ is the start time of job $k$ on machine $g$, $C^L_t$ is the completion time of job $r$ on machine $i$. In this example, $S^L_t < C^L_t$, so we have $t_1 = C^L_t$. Similarly, we have $t_2 = C^R_t$, which is the completion time of job $j$ on machine $f$. Then within $[t_1, t_2)$, the total power consumption in the flow shop is constant.

![Figure 4.1: Gantt chart for Example 4.1.](image)

Inspired by this observation, we propose mixed integer programs with binary variables for Problem PFSPP-DU that keep track of jobs that are running concurrently
on any two different machines only at job start and completion times. First, we propose a mixed integer program in Section 4.1 inspired by Manne (1960), which we will call the disjunctive formulation. In Section 4.2, we propose another mixed integer program inspired by Wagner (1959) and Lasserre and Queyranne (1992), which we will call the assignment and positional formulation (or AP formulation for short).

4.1 Disjunctive formulation

In this section, we propose a mixed integer program inspired by Manne’s (1960) model. We define the following decision variables:

- $C_{\text{max}}$ is the makespan of the schedule;
- $C_{ij}$ is the completion time of job $j$ on machine $i$;
- $S_{ij}$ is the start time of job $j$ on machine $i$;
- $\delta_{jk}$ is equal to 1 if job $j$ precedes job $k$, and 0 otherwise;
- $x_{ij,s}$ is equal to 1 if job $j$ is processed on machine $i$ with speed $s$, and 0 otherwise;
- $u_{hkij}$ is equal to 1 if the start time of job $k$ on machine $h$ is less than or equal to the start time of job $j$ on machine $i$ (in other words, $S_{hk} \leq S_{ij}$), and 0 otherwise;
- $v_{hkij}$ is equal to 1 if the completion time of job $k$ on machine $h$ is greater than the start time of job $j$ on machine $i$ (in other words, $C_{hk} > S_{ij}$), and 0 otherwise;
- $y_{hkij}$ is equal to 1 if the start time of job $j$ on machine $i$ occurs during the processing of job $k$ on machine $h$ (in other words, $S_{hk} \leq S_{ij} < C_{hk}$), and 0 otherwise;
- $z_{hk}\text{sij}$ is equal to 1 if job $k$ is processed on machine $h$ with speed $s$, and starts while job $j$ is running on machine $i$ (in other words, if $x_{hk} = 1$ and $y_{hkij} = 1$), and 0 otherwise.
We call the binary decision variables $u, v, y$ and $z$ the *concurrent job variables*. Let $M$ be an upper bound on the makespan of an optimal schedule. For simplicity, we let $M = \sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{J}} p_{ij}$. Problem PFSP-DU can be modeled as follows:

\[
\begin{align*}
\text{minimize} & \quad C_{\max} \\
\text{subject to} & \quad C_{\max} \geq C_{mj} \quad \text{for } j \in \mathcal{J}; \\
& \quad C_{1j} \geq \sum_{s \in \mathcal{S}} p_{1js} x_{1js} \quad \text{for } j \in \mathcal{J}; \\
& \quad C_{ij} \geq C_{i-1,j} + \sum_{s \in \mathcal{S}} p_{ijs} x_{ijs} \quad \text{for } i \in \mathcal{M}\setminus\{1\}; j \in \mathcal{J}; \\
& \quad C_{ik} \geq C_{ij} + \sum_{s \in \mathcal{S}} p_{iks} x_{iks} - M \delta_{jk} \quad \text{for } i \in \mathcal{M}; j, k \in \mathcal{J}, k > j; \\
& \quad C_{ik} \geq C_{ij} + \sum_{s \in \mathcal{S}} p_{iks} x_{iks} - M(1 - \delta_{jk}) \quad \text{for } i \in \mathcal{M}; j, k \in \mathcal{J}, k > j; \\
& \quad C_{ij} = S_{ij} + \sum_{s \in \mathcal{S}} p_{ijs} x_{ijs} \quad \text{for } i \in \mathcal{M}; j \in \mathcal{J}; \\
& \quad S_{ij} - S_{hk} \leq Mu_{hki} - 1 \quad \text{for } i, h \in \mathcal{M}; j, k \in \mathcal{J}; \\
& \quad S_{hk} - S_{ij} \leq M(1 - u_{hki}) \quad \text{for } i, h \in \mathcal{M}; j, k \in \mathcal{J}; \\
& \quad C_{hk} - S_{ij} \leq Mv_{hki} \quad \text{for } i, h \in \mathcal{M}; j, k \in \mathcal{J}; \\
& \quad S_{ij} - C_{hk} \leq M(1 - v_{hki}) - 1 \quad \text{for } i, h \in \mathcal{M}; j, k \in \mathcal{J}; \\
& \quad u_{hki} + v_{hki} = 1 + y_{hki} \quad \text{for } i, h \in \mathcal{M}; j, k \in \mathcal{J}; \\
& \quad x_{hks} + y_{hki} \leq 1 + z_{hksi} \quad \text{for } i, h \in \mathcal{M}; j, k \in \mathcal{J}; s \in \mathcal{S}; \\
& \quad \sum_{s \in \mathcal{S}} x_{ijs} = 1 \quad \text{for } i \in \mathcal{M}; j \in \mathcal{J}; \\
& \quad \delta_{jk} + \delta_{kj} = 1 \quad \text{for } j, k \in \mathcal{J}, j \neq k; \\
& \quad \delta_{jk} + \delta_{kl} + \delta_{lj} \leq 2 \quad \text{for } j, k, l \in \mathcal{J}, j \neq k \neq l; \\
& \quad \sum_{s \in \mathcal{S}} q_{ijs} x_{ijs} + \sum_{h \in \mathcal{M}, h \neq i} \sum_{k \in \mathcal{J}} \sum_{s \in \mathcal{S}} q_{hks} z_{hksi} \leq Q_{\max} \quad \text{for } i \in \mathcal{M}; j \in \mathcal{J}; \\
& \quad x_{ijs}, \delta_{jk}, u_{hki}, v_{hki}, y_{hki}, z_{hksi} \in \{0, 1\} \quad \text{for } i, h \in \mathcal{M}; j, k \in \mathcal{J}; s \in \mathcal{S}. 
\end{align*}
\]
The objective (4.1a) and the constraints (4.1b) ensure that the makespan is equal to the completion time of the last job processed on machine $m$. Constraints (4.1c)-(4.1f) ensure that the completion times are consistent with a flow shop. In particular, constraints (4.1e)-(4.1f) are the disjunctive constraints between any two jobs $j$ and $k$: they ensure that job $j$ is processed before job $k$ or vice versa. Constraints (4.1g) ensure that jobs are processed nonpreemptively. Constraints (4.1h)-(4.1m) ensure that the concurrent job variables $u, v, y$ and $z$ take their intended values. Constraints (4.1n) indicate that each job can be processed on any given machine with exactly one speed. Constraints (4.1o)-(4.1p) ensure that the jobs are processed in the same order on every machine. Finally, constraints (4.1q) ensure that at any time, the total power consumption across machines does not exceed the threshold $Q_{\text{max}}$.

4.2 Assignment and positional formulation

Next, we give another formulation of Problem PFSPP-DU, inspired by the models proposed by Wagner (1959) and Lasserre and Queyranne (1992), which use binary variables to directly assign jobs to positions in a permutation. A variant of this model was proposed in Fang et al. (2011b). We define the following decision variables:

- $C_{\text{max}}$ is the makespan of the schedule;
- $C_{ij}$ is the completion time of the $j$th job processed on machine $i$ (note that “$j$th job” refers to the $j$th position, not job $j$);
- $S_{ij}$ is the start time of the $j$th job processed on machine $i$;
- $x_{ijks}$ is equal to 1 if job $j$ is the $k$th job processed on machine $i$ with speed $s$, and 0 otherwise;
- $u_{hkij}$ is equal to 1 if the start time of the $k$th job processed on machine $h$ is less than or equal to the start time of the $j$th job processed on machine $i$ (in other words, $S_{hk} \leq S_{ij}$), and 0 otherwise;
• \(v_{hki} \) is equal to 1 if the completion time of the \( k \)th job processed on machine \( h \) is greater than the start time of the \( j \)th job processed on machine \( i \) (in other words, \( C_{hk} > S_{ij} \)), and 0 otherwise;

• \( y_{hki} \) is equal to 1 if the start time of the \( j \)th job processed on machine \( i \) occurs during the processing of the \( k \)th job on machine \( h \) (in other words, \( S_{hk} \leq S_{ij} < C_{hk} \)), and 0 otherwise;

• \( z_{hkl} \) is equal to 1 if job \( l \) is the \( k \)th job processed on machine \( h \) with speed \( s \), and starts while the \( j \)th job is running on machine \( i \) (in other words, if \( x_{hlks} = 1 \) and \( y_{hki} = 1 \)), and 0 otherwise.

As with the disjunctive formulation, we call the decision variables \( u, v, y \) and \( z \) the concurrent job variables.

### 4.2.1 Lower and upper bounds for start and completion time decision variables

For the decision variables representing start and completion times, we can obtain simple lower bounds and upper bounds as follows. Let \( \Omega_{ij} \) be the set of jobs with the smallest \( j \) values of \( \{p_{ikd} : k \in J\} \), and let \( \Delta_j \) be the set that contains the jobs with the largest \( j \) values of \( \{\sum_{i \in \mathcal{M}} p_{ik1} : k \in J\} \). Then, we have the following:

\[
S_{ij} \geq \min_{k \in J} \left\{ \sum_{h=1}^{i-1} p_{hkd} \right\} + \sum_{k \in \Omega_{i,j-1}} p_{ikd} \triangleq S_{ij} \quad \text{for all } i \in \mathcal{M}, j \in \{1, \ldots, n\},
\]

\[
C_{ij} \leq \sum_{k \in \Delta_{j-1}} \sum_{i \in \mathcal{M}} p_{ik1} + \max_{k \in J \setminus \Delta_{j-1}} \left\{ \sum_{h=1}^{i} p_{hkl} \right\} \triangleq C_{ij} \quad \text{for all } i \in \mathcal{M}, j \in \{1, \ldots, n\}.
\]

Clearly, the upper bound \( \overline{C}_{mn} \) for \( C_{mn} \) is also an upper bound for the makespan. For simplicity’s sake, we define \( \overline{S}_{ij} = \overline{C}_{ij} \) and \( \overline{C}_{ij} = \overline{S}_{ij} \) for all \( i \in \mathcal{M} \) and \( j = 1, \ldots, n \).
4.2.2 Basic AP formulation

Using the above lower and upper bounds, we can formulate Problem PFSP-DU as follows:

\[
\begin{align*}
\text{minimize} & \quad C_{\text{max}} & \quad (4.4a) \\
\text{subject to} & \\
C_{\text{max}} & \geq C_{\text{mn}} & \quad (4.4b) \\
C_{ik} & \geq C_{i-1,k} + \sum_{j \in J, s \in S} p_{ij} x_{ij1s} & \quad \text{for } i \in M \setminus \{1\}, k \in \{1, 2, \ldots, n\}; \quad (4.4d) \\
C_{ik} & \geq C_{i,k-1} + \sum_{j \in J, s \in S} p_{ij} x_{ijks} & \quad \text{for } i \in M, k \in \{2, \ldots, n\}; \quad (4.4e) \\
C_{ik} & = S_{ik} + \sum_{j \in J, s \in S} p_{ij} x_{ijks} & \quad \text{for } i \in M; k \in \{1, 2, \ldots, n\}; \quad (4.4f) \\
S_{ij} - S_{hk} & \leq (\bar{S}_{ij} - \bar{S}_{hk}) u_{hkij} - 1 & \quad \text{for } i, h \in M; j, k \in \{1, 2, \ldots, n\}; \quad (4.4g) \\
S_{hk} - S_{ij} & \leq (\bar{S}_{hk} - \bar{S}_{ij})(1 - u_{hkij}) & \quad \text{for } i, h \in M; j, k \in \{1, 2, \ldots, n\}; \quad (4.4h) \\
C_{hk} - C_{ij} & \leq (\bar{C}_{hk} - \bar{C}_{ij}) v_{hkij} & \quad \text{for } i, h \in M; j, k \in \{1, 2, \ldots, n\}; \quad (4.4i) \\
S_{ij} - C_{hk} & \leq (\bar{S}_{ij} - \bar{C}_{hk})(1 - v_{hkij}) - 1 & \quad \text{for } i, h \in M; j, k \in \{1, 2, \ldots, n\}; \quad (4.4j) \\
 u_{hkij} + v_{hkij} & = 1 + y_{hkij} & \quad \text{for } i, h \in M; j, k \in \{1, 2, \ldots, n\}; \quad (4.4k) \\
x_{hlks} + y_{hkij} & \leq 1 + z_{hlksij} & \quad \text{for } i, h \in M; j, k, l \in \{1, 2, \ldots, n\}; s \in S; \quad (4.4l) \\
y_{hkij} & \geq u_{hkij} & \quad \text{for } i, h \in M; j, k \in \{1, 2, \ldots, n\}; \quad (4.4m) \\
y_{hkij} & \geq v_{hkij} & \quad \text{for } i, h \in M; j, k \in \{1, 2, \ldots, n\}; \quad (4.4n) \\
\sum_{k=1}^{n} \sum_{s \in S} x_{ijks} & = 1 & \quad \text{for } i \in M; j \in J; \quad (4.4o) \\
\sum_{j \in J} \sum_{s \in S} x_{ijks} & = 1 & \quad \text{for } i \in M; k \in \{1, 2, \ldots, n\}; \quad (4.4p)
\end{align*}
\]
\[ \sum_{s \in \mathcal{S}} x_{ijks} = \sum_{s \in \mathcal{S}} x_{hjks} \quad \text{for } i, h \in \mathcal{M}; j, k \in \{1, 2, \ldots, n\}; \quad (4.4q) \]

\[ \sum_{h \in \mathcal{M}, h \neq i} \sum_{l \in \mathcal{J}} \sum_{t=1}^{n} \sum_{s \in \mathcal{S}} q_{hls} z_{hlrsik} \]

\[ + \sum_{j \in \mathcal{J}} \sum_{s \in \mathcal{S}} q_{ij} x_{ijks} \leq Q_{\text{max}} \quad \text{for } i \in \mathcal{M}; k \in \{1, 2, \ldots, n\}; \quad (4.4r) \]

\[ x_{ijks}, u_{hkij}, v_{hkij}, y_{hkij}, z_{hlkisj} \in \{0, 1\} \quad \text{for } i, h \in \mathcal{M}; j, l, k \in \{1, 2, \ldots, n\}; s \in \mathcal{S}. \quad (4.4s) \]

The objective (4.4a) and the constraint (4.4b) ensure that the makespan of the schedule is equal to the completion time of the last job on the last machine. Constraints (4.4c)-(4.4e) ensure that the completion times are consistent with a flow shop. Constraints (4.4f) ensure that jobs are processed nonpreemptively. Constraints (4.4g)-(4.4n) ensure that the concurrent job variables \( u, v, y \) and \( z \) take their intended values. Constraints (4.4o) ensure that on each machine each job is processed with exactly one speed and one position. Constraints (4.4p) ensure that each position is assigned with exactly one job and one speed. Constraints (4.4q) ensure that the jobs are processed in the same order on each machine. Finally, constraints (4.4r) ensure that at any time, the total power consumption across machines is at most \( Q_{\text{max}} \). We call the above model (4.4) the basic AP formulation.

### 4.2.3 Strengthening the basic AP formulation: concurrent job valid inequalities

Recall that the variables \( u, v, y, z \) are related to the jobs running concurrently at job start and completion times. In this subsection, we show how to strengthen the basic AP formulation by giving valid inequalities based on the definitions of these variables. We call these inequalities the concurrent job valid inequalities.
Theorem 4.1  The following inequalities are valid for the integer program (4.4):

\[ \sum_{r=k+1}^{n} u_{hrij} \leq (n-k)u_{hki} \quad \text{for } i, h \in \mathcal{M}; j, k \in \{1,2,\ldots,n\}. \]  

(4.5a)

Proof  Fix \( h, k, i, j \). If \( u_{hki} = 0 \), i.e. \( S_{hk} > S_{ij} \), then for each \( r = k+1, \ldots, n \), we must also have \( S_{hr} > S_{ij} \) in any feasible schedule, and so by the definition of the variables \( u \), we have \( u_{hrij} = 0 \). On the other hand, if \( u_{hki} = 1 \), the left hand side of inequality (4.5a) is at most \( n-k \), since \( u_{hrij} \) for \( r = k+1, \ldots, n \) are binary variables. Therefore, the above inequality is valid.

Theorem 4.2  The following inequalities are valid for the integer program (4.4):

\[ \sum_{r=1}^{k-1} v_{hrij} \leq (k-1)v_{hki} \quad \text{for } i, h \in \mathcal{M}; j, k \in \{1,2,\ldots,n\} \).  

(4.5b)

Proof  Fix \( h, k, i, j \). If \( v_{hki} = 0 \), i.e. \( C_{hk} \leq S_{ij} \), then for each \( r = 1,2,\ldots,k-1 \), we must also have \( C_{hr} \leq S_{ij} \) in any feasible schedule, and so by the definition of variables \( v \), we have \( v_{hrij} = 0 \). On the other hand, if \( v_{hki} = 1 \), the left hand side of inequality (4.5b) is at most \( k-1 \), since \( v_{hrij} \) for \( r = 1,2,\ldots,k-1 \) are binary variables. Therefore, the above inequality is valid.

Theorem 4.3  The following inequalities are valid for the integer program (4.4):

\[ y_{hki} = \sum_{l \in \mathcal{J}} \sum_{s \in \mathcal{S}} z_{hlksij} \quad \text{for } i, h \in \mathcal{M}; j, k \in \{1,2,\ldots,n\}. \]  

(4.5c)

Proof  Fix \( h, k, i, j \). If \( y_{hki} = 0 \), then by the definition of variables \( z \), we have \( z_{hlksij} = 0 \) for all \( l \) and \( s \). Now suppose \( y_{hki} = 1 \). From constraints (4.4p) we have \( \sum_{l \in \mathcal{J}} \sum_{s \in \mathcal{S}} x_{hlks} = 1 \). Without loss of generality, suppose job \( r \) is assigned to position \( k \) on machine \( h \) with speed \( s \); i.e., \( x_{hrks} = 1 \). Then by the definition of variables \( z \), we have \( z_{hrksij} = 1 \) and \( z_{hlksij} = 0 \) for any other job \( l \neq r \), and so \( \sum_{l \in \mathcal{J}} \sum_{s \in \mathcal{S}} z_{hlksij} = 1 \).
Theorem 4.4 The following inequalities are valid for the integer program (4.4):

\[ \sum_{k=1}^{n} y_{hkij} \leq 1 \quad \text{for } i, h \in \mathcal{M}; j \in \{1, 2, \ldots, n\}. \] (4.5d)

Proof For each position \( j \) on machine \( i \), there exists at most one position \( k \) on machine \( h \) that satisfies \( S_{hk} \leq S_{ij} < C_{hk} \), since at most one job is processed in each position. \( \blacksquare \)

Theorem 4.5 The following inequalities are valid for the integer program (4.4):

\[ u_{hkij} + u_{ijhk} \leq \frac{1}{2} (y_{hkij} + y_{ijhk}) + 1 \leq v_{hkij} + v_{ijhk} \quad \text{for } i, h \in \mathcal{M}; j, k \in \{1, 2, \ldots, n\}. \] (4.5e)

Proof Fix \( h, k, i, j \). Suppose \( y_{hkij} + y_{ijhk} = 0 \), i.e., \( y_{hkij} = y_{ijhk} = 0 \), or \( S_{hk} \geq C_{ij} \) or \( S_{ij} \geq C_{hk} \). Without loss of generality, assume \( S_{hk} \geq C_{ij} \), which implies \( C_{hk} \geq S_{hk} \geq C_{ij} \geq S_{ij} \). Then, by definition we have \( u_{hkij} = 0, u_{ijhk} = 1, v_{hkij} = 1, \) and \( v_{ijhk} = 0 \), and so the inequality (4.5e) holds. Now suppose \( y_{hkij} + y_{ijhk} = 1 \). Without loss of generality, assume \( y_{hkij} = 1 \) and \( y_{ijhk} = 0 \). Then we have \( S_{hk} \leq S_{ij} < C_{hk} \). By definition we have \( u_{hkij} = 1, u_{ijhk} = 0, v_{hkij} = v_{ijhk} = 1, \) and so the inequality (4.5e) also holds. Finally, if \( y_{hkij} + y_{ijhk} = 2 \), i.e., \( y_{hkij} = y_{ijhk} = 1 \), or \( S_{hk} = S_{ij} \), then by definition we have \( u_{hkij} = u_{ijhk} = 1 \) and \( v_{hkij} = v_{ijhk} = 1 \), and so inequality (4.5e) still holds. \( \blacksquare \)

Theorem 4.6 For each \( i, h \in \mathcal{M} \) and \( j \in \{1, 2, \ldots, n\} \), the following inequalities are valid for the integer program (4.4):

\[ u_{h,k+1,ij} + v_{h,k-1,ij} \leq 1 - y_{hkij} \quad \text{for } k \in \{2, \ldots, n - 1\}. \] (4.5f)

Proof Fix \( h, k, i, j \). If \( y_{hkij} = 1 \), i.e., \( S_{hk} \leq S_{ij} < C_{hk} \), then we have \( S_{h,k+1} > S_{ij} \) and \( C_{h,k-1} \leq S_{ij} \), or in other words, \( u_{h,k+1,ij} = v_{h,k-1,ij} = 0 \). So in this case, inequality (4.5f) holds. Otherwise, because \( C_{h,k-1} < S_{h,k+1} \), we have that \( S_{h,k+1} \leq S_{ij} \) and \( C_{h,k-1} > S_{ij} \)
cannot be satisfied simultaneously, or in other words, \( u_{h,k+1,ij} + v_{h,k-1,ij} \leq 1 \). So in this case as well, inequality (4.5f) holds.

**Theorem 4.7** For each \( i, h \in M \) and \( j \in \{2, \ldots, n-1\} \), the following inequalities are valid for the integer program (4.4):

\[
 u_{hki,j+1} + v_{hki,j-1} \geq 1 + y_{hkij} \quad \text{for } k \in \{1, 2, \ldots, n\}. \tag{4.5g}
\]

**Proof** Fix \( h, k, i, j \). If \( y_{hkij} = 1 \), i.e., \( S_{hk} \leq S_{ij} < C_{hk} \), then we have \( S_{hk} < S_{i,j+1} \) and \( C_{hk} > S_{i,j-1} \). In other words, \( u_{hki,j+1} = v_{hki,j-1} = 1 \), and so in this case, inequality (4.5g) holds. Otherwise, we have that either \( C_{i,j-1} \leq S_{hk} \) or \( C_{hk} \leq S_{i,j+1} \), which implies \( u_{hki,j+1} + v_{hki,j-1} \geq 1 \). So in this case as well, inequality (4.5g) holds.

### 4.2.4 Strengthening the basic AP formulation: nondelay valid inequalities

A feasible schedule is called *nondelay* if no machine is idle when there exists a job that can be processed without violating the threshold \( Q_{\text{max}} \) on the total power consumption across all machines. It turns out for Problem PFSPP-DU, we can restrict our attention to nondelay schedules without loss of generality.

**Theorem 4.8** For Problem PFSPP-DU, there always exists an optimal schedule that is nondelay.

**Proof** Suppose schedule \( \sigma_1 \) is an optimal schedule that is not nondelay for Problem PFSPP-DU. Let \( C_{\sigma_1} \) be the makespan of \( \sigma_1 \). Suppose machine \( i \) is idle when job \( j \) is available to be processed in schedule \( \sigma_1 \). Then we process all the other jobs the same way, and process job \( j \) earlier with the same speed as in schedule \( \sigma_1 \), starting at the earliest time after job \( j \) is completed on machine \( i-1 \) at which scheduling job \( j \) on machine \( i \) does not violate the power consumption constraints. Denote this new schedule as \( \sigma_2 \) with makespan \( C_{\sigma_2} \). If the completion time of job \( j \) on machine \( i \) in \( \sigma_1 \)
is not the unique value that determines the makespan, then we still have $C_{\sigma_2} = C_{\sigma_1}$. Now suppose $j$ is the only job that attains the makespan $C_{\sigma_1}$ in schedule $\sigma_1$. Then by processing job $j$ earlier on machine $i$, we obtain $C_{\sigma_2} < C_{\sigma_1}$, which contradicts the optimality of schedule $\sigma_1$.

Based on Theorem 4.8, we can add constraints to the basic AP formulation (4.4) that require schedules to be nondelay. It is difficult to obtain such inequalities for the general PFSP problem. However, when the flow shop has two machines, i.e., $\mathcal{M} = \{1, 2\}$, then we have the following **nondelay valid inequalities** (4.6).

**Theorem 4.9** Suppose $\mathcal{M} = \{1, 2\}$. For each $j \in \{2, 3, \ldots, n\}$, the following inequalities are valid for the integer program (4.4):

\[
\begin{align*}
    y_{2k1j} + 1 - v_{1,j-1,2k} &\leq u_{2k1j} + u_{1j2k} & \text{for } k \in \{1, 2, \ldots, j-1\}, \\
    y_{1k2j} + 1 - v_{2,j-1,1k} &\leq u_{1k2j} + u_{2j1k} & \text{for } k \in \{j+1, \ldots, n\}.
\end{align*}
\] (4.6a)

**Proof** Fix $j, k$. If $y_{2k1j} = 1$, i.e., $S_{2k} \leq S_{1j} < C_{2k}$, and $v_{1,j-1,2k} = 0$, i.e., $C_{1,j-1} \leq S_{2k}$, then because there always exists an optimal nondelay schedule, we can start the job in the $j$th position on machine 1 simultaneously with the job in the $k$th position on machine 2, i.e., $S_{2k} = S_{1j}$.

\[
\begin{array}{c}
\text{machine 1} \\
\text{machine 2}
\end{array}
\begin{array}{c|c|c|c|c}
\cdots & j-1 & \ast & j & \cdots \\
\cdots & k & \cdots
\end{array}
\]

In other words, $u_{2k1j} = u_{1j2k} = 1$. So in this case the inequality (4.6a) holds. Otherwise, if $y_{2k1j} = 0$, because $u_{2k1j} + u_{1j2k} \geq 1$ and $v_{1,j-1,2k} \geq 0$, the inequality (4.6a) also holds.

Reversing the roles of machines 1 and 2, we similarly obtain valid inequalities (4.6b).
Theorem 4.10 Suppose $\mathcal{M} = \{1, 2\}$. The following inequalities are valid for the integer program ($4.4$):

$$y_{1k21} \leq u_{211k} \quad \text{for } k \in \{2, 3, \ldots, n\}. \tag{4.6c}$$

**Proof** Fix $k$. If $y_{1k21} = 1$, i.e. $S_{1k} \leq S_{21} < C_{1k}$, then we can start the job in the first position on machine 2 simultaneously with the job in the $k$th position on machine 1, that is $S_{1k} = S_{21}$.

In other words, $u_{211k} = 1$, and so inequality (4.6c) holds.

Theorem 4.11 Suppose $\mathcal{M} = \{1, 2\}$. For each $j \in \{2, 3, \ldots, n\}$, the following inequalities are valid for the integer program ($4.4$):

$$S_{1j} - C_{1,j-1} \leq (S_{1j} - C_{1,j-1})(2 - v_{1,j-1,2k} - y_{2k1j}) \quad \text{for } k \in \{1, 2, \ldots, j-2\}, \tag{4.6d}$$

$$S_{2j} - C_{2,j-1} \leq (S_{2j} - C_{2,j-1})(2 - v_{2,j-1,1k} - y_{1k2j}) \quad \text{for } k \in \{j+1, \ldots, n\}. \tag{4.6e}$$

**Proof** Fix $j, k$. Suppose $y_{2k1j} = 1$ and $v_{1,j-1,2k} = 1$, i.e., $S_{2k} \leq S_{1j} < C_{2k}$ and $C_{1,j-1} > S_{2k}$.

Then because there always exists an optimal nondelay schedule, we can process the job in the $j$th position immediately after the completion time of the job in the $(j-1)$th position on machine 1; that is, $S_{1j} = C_{1,j-1}$. So the inequality (4.6d) holds. Now suppose $y_{2k1j}$ and $v_{1,j-1,2k}$ are not both equal to 1. Then the right side is at least $S_{1j} - C_{1,j-1}$, and so the inequality (4.6d) still holds.

Reversing the roles of machines 1 and 2, we similarly obtain valid inequalities (4.6e).
We call the model that combines the basic AP formulation with the concurrent job valid inequalities (4.5) (and the nondelay valid inequalities (4.6) when $\mathcal{M} = \{1, 2\}$) the *enhanced AP formulation*.

### 4.3 Experimental study

#### 4.3.1 Computational environment

In this section, we compare the performance of the different formulations presented in Section 4.1 and Section 4.2, with respect to both computation time and solution quality. We used Gurobi Optimizer 4.5 to solve the mixed integer programs on a computer with two 2.5 GHz Quad-Core AMD 2380 processors and 32GB of RAM running the Linux operating system.

![Cast iron plates with slots.](image)

To conduct our experiments, we considered a hypothetical flow shop scheduling problem arising from the manufacture cast iron plates with slots (Figure 4.2). The plates manufactured in this flow shop can have three different lengths, two different depths of milling on the surface, three different numbers of slots, and two different depths of slots. In other words, there are $3 \times 2 \times 3 \times 2 = 36$ different types of parts. There are two types of machines with different operations: face milling and profile milling. We consider two different cases: in the first case, each plate must have the two operations on one side; in the second case, each plate must have the two operations on two sides. We can view these two different cases as two different flow shop problems with 2 and 4 machines, respectively; that is, $\mathcal{M} = \{1, 2\}$ or $\mathcal{M} = \{1, 2, 3, 4\}$. There are
five available cutting speeds on each machine; that is, \( d = 5 \) and \( S = \{ s_1, s_2, s_3, s_4, s_5 \} \).

We also consider a special case in which we can only use each machine’s slowest and fastest speeds; that is, \( d = 2 \) and \( S = \{ s_1, s_2 \} \). Cutting speeds for both face milling and profile milling are chosen to be within the range recommended by the Machinery’s Handbook (Oberg et al. 2008). For a job processed on one of the machines, the processing time \( p_{ijs} \) and power consumption \( q_{ijs} \) can be calculated in accordance with the cutting speeds (Fang et al. 2011b). In the instances we study, the processing times \( p_{ijs} \) are in the interval \([4, 22]\) (in minutes), and the power consumption values \( q_{ijs} \) are in the interval \([4, 9]\) (in kW). For this study, we consider instances in which the number of jobs \( n \) is 8, 12, 16, or 20.

We generate 10 different settings for each combination of \((m, n, d)\), by randomly sampling \( n \) jobs with replacement from the 36 job types. Let \((m, n, d, k)\) denote the \( k \)th setting that has \( m \) machines, \( n \) jobs and \( d \) speeds. In summary, the family of settings we use in our experiment is

\[
\{ (2, n, 2, k), (2, n, 5, k), (4, n, 2, k) : n \in \{8, 12, 16, 20\}, k \in \{1, 2, \ldots, 10\} \}.
\]

For each setting \((m, n, d, k)\), we let \( Q = \max_{i \in M,j \in J} \{q_{ij1}\} \), \( \overline{Q} = \sum_{i \in M} \max_{j \in J} \{q_{ijd}\} \).

Note that when \( Q_{\text{max}} < Q \), there is no feasible schedule, and when \( Q_{\text{max}} > \overline{Q} \), all jobs can be processed concurrently at their maximum speed. We call \([Q, \overline{Q}]\) the power consumption range for each instance, which we divide into 9 subintervals with same length. For each setting \((m, n, d, k)\), we solve the corresponding mixed integer program using the 10 endpoints of these subintervals as the threshold \( Q_{\text{max}} \) on the total power consumption at any time instant. This way, for each combination of \( m, n \) and \( d \), we will test 10 settings with 10 different peak power consumption thresholds, or 100 instances of the problem. We set a 30 minute time limit on each instance.
4.3.2 Two heuristic algorithms for finding feasible schedules

For an instance \((m, n, d, k)\) with threshold \(Q_{\text{max}}\) on the peak power consumption, let \(s^*_ij\) be the maximum possible speed at which job \(j\) can be processed on machine \(i\) individually without violating the power consumption threshold; i.e., \(s^*_ij = \max\{s \in S : q_{ijs} \leq Q_{\text{max}}\}\). Suppose we fix a permutation of the job set. Algorithm 4.3.1 is a straightforward algorithm that finds a feasible schedule by processing each job \(j\) on machine \(i\) at speed \(s^*_ij\) without overlap – that is, with no other concurrently running jobs – according to this fixed permutation of the job set.

**Algorithm 4.3.1** Maximum possible speed, fix permutation, no overlap

**Require:** \(p_{ijs}, q_{ij}\) for \(i \in M, j \in J, s \in S, \) permutation \((1, \ldots, n)\) of \(J\).

1: for \(j = 1\) to \(n\) do
2: \hspace{1em} for \(i = 1\) to \(m\) do
3: \hspace{2em} calculate \(s^*_ij\).
4: \hspace{2em} schedule job \(j\) on machine \(i\) at speed \(s^*_ij\) without overlap.
5: \hspace{1em} end for
6: end for

For example, when Algorithm 4.3.1 is applied to a two-machine flow shop, in which each job is processed with its maximum possible speed without overlap, the Gantt chart of the resulting schedule looks like this:

```
  machine 1 | 1  | 2  | ... | n |
  machine 2 | 1  | 2  | ... | n |
```

Another simple method to generate feasible solutions is to process jobs as early as possible at their minimum possible speeds, while respecting the power consumption constraints. Algorithm 4.3.2 gives psuedocode for this simple method. As mentioned above (e.g., Example 4.1), we only need to keep track of the jobs that are running concurrently at start or completion times. Let \(\tilde{T}\) be the sorted list of all the start and completion times of jobs, and let \(\tilde{T}[i]\) be the \(i\)th component in \(\tilde{T}\).

For example, suppose that the following is a Gantt chart for the schedule we obtain using Algorithm 4.3.2 for the first 3 jobs of a two-machine instance:
Algorithm 4.3.2 Slowest speed, fix permutation, as early as possible

Require: \( p_{ijs}, q_{ijs} \) for \( i \in \mathcal{M}, j \in \mathcal{J}, s \in \mathcal{S} \), permutation \((1, \ldots, n)\) of \( \mathcal{J} \).

1: for \( j = 1 \) to \( n \) do 
2: \hspace{1em} for \( i = 1 \) to \( m \) do 
3: \hspace{2em} schedule job \( j \) on machine \( i \) at its earliest possible time \( \tilde{T}[l] \).
4: \hspace{1em} while power consumption constraints are violated do 
5: \hspace{2em} \( l = l + 1 \)
6: \hspace{2em} schedule job \( j \) on machine \( i \) at the next available time \( \tilde{T}[l] \).
7: \hspace{1em} end while 
8: update \( \tilde{T} \).
9: end for 
10: end for

Here, \( \tilde{T} = \{t_1, \ldots, t_7\} \). Next, Algorithm 4.3.2 attempts to schedule job 4 on machine 1 at its earliest possible time \( t_4 \). If the power consumption threshold is violated when job 4 is processed on machine 1 at time \( t_4 \), then the algorithm tries to schedule job 4 on machine 1 at the next possible time \( t_5 \). If the power consumption threshold is satisfied, then the algorithm schedules job 4 on machine 1 at \( t_5 \) and updates the list of times \( \tilde{T} \). The algorithm proceeds like this until job 4 is scheduled on machine 1.

The above two heuristic algorithms provide us a quick way to obtain an upper bound on optimal solution. Unfortunately, we find that these upper bounds can be quite loose in most of our computational results, since they only consider the maximum and minimum speeds at which jobs can be processed. When jobs can be processed with a wide range of speeds, we do not expect that these heuristics perform well when compared to an exact algorithm.

4.3.3 Experiment 1: makespan \( (C_{\text{max}}) \) vs. power consumption \( (Q_{\text{max}}) \)

From Assumption 3.1, we know that when a job is processed at a higher speed, its processing time decreases, while its power consumption increases. Based on this
assumption, one expects a significant trade-off between makespan and peak power consumption. To achieve the shortest possible makespan, jobs should be processed at their highest possible speeds simultaneously without idle time, which leads to high peak power consumption. On the other hand, if the objective is to minimize peak power consumption, jobs should be processed at their lowest speeds and the machines should be operated without overlap. Figure 4.3 shows an example of the relationship between makespan and power consumption for a setting with $m = 2$, $n = 8$, and $d = 2$.

![Figure 4.3.: Trade-off between makespan and peak power consumption.](image)

Figure 4.3.: Trade-off between makespan and peak power consumption.

We also observe that the running times of all the formulations are considerably higher for instances with intermediate values of $Q_{\text{max}}$. Figure 4.4 shows an example of the relationship between $Q_{\text{max}}$ and running time for the same setting with $m = 2$, $n = 8$ and $d = 2$, using the enhanced AP formulation. This seems correct, intuitively: for extreme values of $Q_{\text{max}}$, the optimal schedule is straightforward to compute, whereas for intermediate values of $Q_{\text{max}}$, the scheduling (and in particular, deciding the processing speeds) is trickier.

As we will see in the subsequent subsections, this pattern is prevalent. Since the running times of the formulations appear to be related to the value of $Q_{\text{max}}$, we divide
the power consumption range into 3 classes. For an instance with power consumption range \([Q, \overline{Q}]\), if \(Q_{\text{max}}\) is less than the first quartile of \([Q, \overline{Q}]\), then we call \(Q_{\text{max}}\) “low”; when \(Q_{\text{max}}\) is greater than the third quartile of \([Q, \overline{Q}]\), then we call \(Q_{\text{max}}\) “high”; otherwise, we call \(Q_{\text{max}}\) “intermediate.” In the following experiments, we will analyze the performance of the mixed integer programs with respect to different classes of \(Q_{\text{max}}\).

4.3.4 Experiment 2: disjunctive vs. basic AP formulation

In order to compare the performance of the different formulations, we use the following measures to assess their performance:

- The number of instances solved to optimality within the 30 minute time limit.
- The average and maximum solution time for these instances solved to optimality.
- The average and maximum speedup factor of the running time for these instances solved to optimality. For any two formulations \(a\) and \(b\), we define the speedup factor (SF) between \(a\) and \(b\) for a given instance as the ratio between the times taken by \(a\) and \(b\) to solve the instance. We only compute a speedup factor when both formulations can solve the instances to optimality within the predetermined time limit.
- The average optimality gap at various time points within the 30 minute time limit. We define the optimality gap as the ratio of the value of the best known feasible solution to the best known lower bound. This measure lets us compare the performance of the different formulations for the instances that do not solve to optimality within the time limit.

These performance measures were also used by Keha et al. (2009) and Unlu and Mason (2010) in their study of the computational performance of mixed integer programming formulations for various scheduling problems.
In this experiment, we compare the performance of the disjunctive and basic AP formulations. We look at a family of instances similarly constructed to the one described in Section 4.3.1, except that we look at settings with \( m = 2 \), \( n \in \{4, 5, 6, 7, 8\} \), and \( d = 2 \). We focus on these smaller instances in this experiment because as we see in Table 4.1 (on page 43), the disjunctive formulation for even moderately sized instances (e.g., \( n = 8 \)) fails to solve within the 30 minute time limit. Table 4.1 shows the average running times for the disjunctive and basic AP formulations, and Table 4.2 shows the average speedup factor between the disjunctive and basic AP formulations.

From Tables 4.1 and 4.2, we can see that the basic AP formulation runs much faster than the disjunctive formulation, especially for instances with larger numbers of jobs. Figure 4.5 graphically shows the average speedup factor between the disjunctive and basic AP formulations. Given these observations, we decided to devote more attention to strengthened versions of the basic AP formulation.

\[
\begin{array}{c}
\text{Number of jobs} \\
4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Speedup factor} \\
0 & 100 & 200 & 300 \\
\end{array}
\]

Figure 4.5.: Average speedup factors between the disjunctive and basic AP formulations.

### 4.3.5 Experiment 3: basic AP vs. enhanced AP formulation

We proposed the concurrent valid inequalities for basic AP formulation in Section 4.2.3, and the nondelay valid inequalities when \( m = 2 \) in Section 4.2.4. We expect that the addition of these inequalities in the enhanced AP formulation will result in
better computational results than the basic AP formulation. As mentioned in Section 4.3.1, we consider instances with 2 machines and 4 machines.

Table 4.3 displays the size of the basic and the enhanced AP formulations for an instance with $m = 2$ and $d = 2$. The enhanced AP formulation has the same number of variables as the basic AP formulation, but the enhanced AP formulation has more constraints and nonzeros than the basic AP formulation, since the concurrent job valid inequalities (and nondelay valid inequalities when $m = 2$) are also included.

Table 4.5 shows the computational results for the instances solved to optimality with low or high values of $Q_{\text{max}}$. From this table, we see that the number of instances solved in the basic AP formulation is about the same as the enhanced AP formulation, except for instances with $m = 4, n = 12, d = 2$, for which only one of the instances was solved to optimality using the basic AP formulation while 3 instances were solved using the enhanced AP formulation. We also see that the average speedup factor in many cases is less than 1, meaning that the basic AP formulation solved faster on average than the enhanced AP formulation. This observation might be explained by the low and high $Q_{\text{max}}$: in Section 4.3.3 we observed that the instances with low or high values of $Q_{\text{max}}$ are “easy.” Given this, one might expect that the running times of these formulations for these instances are mainly based on the size of the formulations, not the difficulty of the scheduling decisions.

Table 4.6 shows the computational results for the instances with intermediate values of $Q_{\text{max}}$. From this table, we see that in most cases, the average speedup factor is larger than 1. In other words, for these instances with intermediate values of $Q_{\text{max}}$, the additional valid inequalities in the enhanced AP formulation help in reducing running times.

When analyzing the data in more detail, we found more evidence that the additional valid inequalities are effective in reducing running times for instances with intermediate $Q_{\text{max}}$. Table 4.4 shows the computational results for instances with intermediate values of $Q_{\text{max}}$ in which $m = 2, n = 8, d = 2$. From Table 4.4, we see that for most instances, the enhanced AP formulation runs faster than the basic AP formulation (i.e.
53 out of 60). On the other hand, for instances in which the basic AP formulation is faster, we see that the average running times for both formulations are much smaller (less than 2 seconds).

These observations suggest something similar to what we observed with the data in Table 4.5. When the problem is easy to solve, the size of formulation is the main factor in the running time, implying that the basic AP formulation should be faster. Otherwise, when the problem is more difficult, the valid inequalities added in the enhanced AP formulation significantly reduce the running time.

We also observed that the majority of instances, especially those with larger \(m, n,\) and \(d,\) did not solve within 30 minutes, even using the enhanced AP formulation. Tables 4.7, 4.8 and 4.9 show the optimality gap at various times for those instances with both AP formulations within the 30 minute time limit. From Tables 4.7, 4.8 and 4.9, we see that the valid inequalities help in reducing the optimality gap of the mixed integer program at various time points in the solution process (especially at the later times). In addition, we see that as the number of jobs increases, the solution quality decreases at any given time, since the increased number of jobs adds to the difficulty of scheduling jobs.

### 4.3.6 Tables

Table 4.1: Comparison of average running time between disjunctive and basic AP formulations.

<table>
<thead>
<tr>
<th>(m)</th>
<th>(d)</th>
<th>(Q_{\text{max}}) Range</th>
<th>formulation</th>
<th>(n = 4)</th>
<th>(n = 5)</th>
<th>(n = 6)</th>
<th>(n = 7)</th>
<th>(n = 8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>low</td>
<td>disjunctive</td>
<td>0.07</td>
<td>0.78</td>
<td>11.81</td>
<td>280.60</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic AP</td>
<td>0.05</td>
<td>0.15</td>
<td>0.45</td>
<td>0.85</td>
<td>1.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>intermediate</td>
<td>disjunctive</td>
<td>0.12</td>
<td>0.51</td>
<td>5.87</td>
<td>127.80</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic AP</td>
<td>0.18</td>
<td>0.31</td>
<td>2.09</td>
<td>17.96</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
<td>disjunctive</td>
<td>0.09</td>
<td>0.16</td>
<td>0.49</td>
<td>2.83</td>
<td>24.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic AP</td>
<td>0.07</td>
<td>0.06</td>
<td>0.11</td>
<td>0.28</td>
<td>0.62</td>
<td></td>
</tr>
</tbody>
</table>

NA: not all instances solved within the time limit.
Table 4.2: Speedup factor between disjunctive and basic AP formulations.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$d$</th>
<th>$Q_{\text{max}}$ Range</th>
<th>$n = 4$</th>
<th>$n = 5$</th>
<th>$n = 6$</th>
<th>$n = 7$</th>
<th>$n = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>low</td>
<td>1.43</td>
<td>3.63</td>
<td>33.95</td>
<td>346.00</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>intermediate</td>
<td>0.76</td>
<td>2.85</td>
<td>16.00</td>
<td>138.00</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
<td>2.57</td>
<td>4.03</td>
<td>4.95</td>
<td>12.03</td>
<td>52.27</td>
</tr>
</tbody>
</table>

NA: not all instances solved within the time limit.

Table 4.3: Initial sizes of the basic and the enhanced AP formulations.

<table>
<thead>
<tr>
<th>$n$</th>
<th>AP formulation</th>
<th># of variables</th>
<th># of constraints</th>
<th># of nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>basic</td>
<td>2721</td>
<td>2874</td>
<td>12001</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>2721</td>
<td>4204</td>
<td>18427</td>
</tr>
<tr>
<td>12</td>
<td>basic</td>
<td>8401</td>
<td>8726</td>
<td>36237</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>8401</td>
<td>11776</td>
<td>54415</td>
</tr>
<tr>
<td>16</td>
<td>basic</td>
<td>19009</td>
<td>19570</td>
<td>80825</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>19009</td>
<td>25044</td>
<td>119523</td>
</tr>
<tr>
<td>20</td>
<td>basic</td>
<td>36081</td>
<td>36942</td>
<td>151909</td>
</tr>
<tr>
<td></td>
<td>enhanced</td>
<td>36081</td>
<td>45544</td>
<td>222199</td>
</tr>
</tbody>
</table>

Table 4.4: Results for instances with intermediate values of $Q_{\text{max}}$ and $m = 2, n = 8, d = 2$.

<table>
<thead>
<tr>
<th>Result type</th>
<th># of instances</th>
<th>formulation</th>
<th>Avg time</th>
<th>Max time</th>
<th>Avg SF</th>
<th>Max SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>enhanced AP is faster</td>
<td>53</td>
<td>basic AP</td>
<td>527.00</td>
<td>1800.00</td>
<td>4.54</td>
<td>8.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced AP</td>
<td>137.23</td>
<td>533.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>basic AP is faster</td>
<td>7</td>
<td>basic AP</td>
<td>0.76</td>
<td>1.54</td>
<td>0.46</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced AP</td>
<td>1.57</td>
<td>2.23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 4.5: Results for solved instances with low or high $Q_{\text{max}}$.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>$m$</th>
<th>$d$</th>
<th>$Q_{\text{max}}$ Range</th>
<th># of instances</th>
<th>$n$</th>
<th># solved</th>
<th>Avg time</th>
<th>Max time</th>
<th>$n$</th>
<th># solved</th>
<th>Avg time</th>
<th>Max time</th>
<th>Avg SF</th>
<th>Max SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic AP</td>
<td>2</td>
<td>2</td>
<td>low</td>
<td>20</td>
<td>8</td>
<td>20</td>
<td>1.10</td>
<td>1.77</td>
<td>20</td>
<td>1.53</td>
<td>2.90</td>
<td>0.81</td>
<td>1.62</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td>20</td>
<td>109.70</td>
<td>208.30</td>
<td>20</td>
<td>54.67</td>
<td>88.64</td>
<td>1.96</td>
<td>3.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>enhanced AP</td>
<td>2</td>
<td>2</td>
<td>low</td>
<td>20</td>
<td>8</td>
<td>20</td>
<td>74.81</td>
<td>763.14</td>
<td>20</td>
<td>25.25</td>
<td>300.69</td>
<td>0.93</td>
<td>4.63</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td>19</td>
<td>29.98</td>
<td>223.78</td>
<td>19</td>
<td>48.90</td>
<td>409.30</td>
<td>0.60</td>
<td>2.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>16</td>
<td>18</td>
<td>118.32</td>
<td>1334.63</td>
<td>18</td>
<td>134.24</td>
<td>792.74</td>
<td>0.51</td>
<td>1.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>20</td>
<td>18</td>
<td>181.11</td>
<td>1200.92</td>
<td>17</td>
<td>297.88</td>
<td>1313.01</td>
<td>0.53</td>
<td>3.34</td>
<td></td>
</tr>
<tr>
<td>basic AP</td>
<td>2</td>
<td>5</td>
<td>low</td>
<td>20</td>
<td>8</td>
<td>20</td>
<td>1.64</td>
<td>2.50</td>
<td>20</td>
<td>2.00</td>
<td>3.89</td>
<td>0.89</td>
<td>1.70</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td>20</td>
<td>160.55</td>
<td>205.62</td>
<td>20</td>
<td>112.15</td>
<td>386.55</td>
<td>1.67</td>
<td>2.77</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>enhanced AP</td>
<td>2</td>
<td>5</td>
<td>low</td>
<td>20</td>
<td>8</td>
<td>20</td>
<td>1.58</td>
<td>11.20</td>
<td>20</td>
<td>1.76</td>
<td>3.51</td>
<td>0.83</td>
<td>3.19</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td>20</td>
<td>8.71</td>
<td>59.62</td>
<td>20</td>
<td>24.05</td>
<td>55.97</td>
<td>0.41</td>
<td>1.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>16</td>
<td>20</td>
<td>19.56</td>
<td>81.00</td>
<td>20</td>
<td>80.74</td>
<td>213.36</td>
<td>0.40</td>
<td>1.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>99.79</td>
<td>514.15</td>
<td>20</td>
<td>249.40</td>
<td>723.71</td>
<td>0.56</td>
<td>4.10</td>
<td></td>
</tr>
<tr>
<td>basic AP</td>
<td>4</td>
<td>2</td>
<td>low</td>
<td>20</td>
<td>8</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>enhanced AP</td>
<td>4</td>
<td>2</td>
<td>low</td>
<td>20</td>
<td>8</td>
<td>19</td>
<td>406.58</td>
<td>1109.00</td>
<td>19</td>
<td>499.11</td>
<td>1151.89</td>
<td>1.02</td>
<td>4.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td>1</td>
<td>1490.82</td>
<td>1490.82</td>
<td>3</td>
<td>1425.87</td>
<td>1532.59</td>
<td>1.20</td>
<td>1.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.6: Results for solved instances with intermediate $Q_{\text{max}}$.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>$m$</th>
<th>$d$</th>
<th>$Q_{\text{max}}$ Range</th>
<th># of instances</th>
<th>$n$</th>
<th># solved</th>
<th>Avg time</th>
<th>Max time</th>
<th>$\text{# solved}$</th>
<th>Avg time</th>
<th>Max time</th>
<th>Avg SF</th>
<th>Max SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic AP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enhanced AP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2$ 2 intermediate</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>59</td>
<td>442.99</td>
<td>1682.51</td>
<td>60</td>
<td>121.41</td>
<td>533.91</td>
<td>4.06</td>
<td>8.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>9</td>
<td>103.63</td>
<td>497.80</td>
<td>9</td>
<td>66.16</td>
<td>210.29</td>
<td>1.87</td>
<td>7.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>8</td>
<td>204.69</td>
<td>1172.95</td>
<td>9</td>
<td>304.31</td>
<td>1452.12</td>
<td>1.57</td>
<td>7.34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>8</td>
<td>355.16</td>
<td>889.05</td>
<td>7</td>
<td>440.15</td>
<td>1311.67</td>
<td>0.81</td>
<td>1.74</td>
</tr>
<tr>
<td>$2$ 5 intermediate</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>38</td>
<td>270.23</td>
<td>1353.97</td>
<td>60</td>
<td>390.02</td>
<td>1591.26</td>
<td>2.11</td>
<td>6.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>17</td>
<td>107.08</td>
<td>226.56</td>
<td>18</td>
<td>207.02</td>
<td>1307.46</td>
<td>0.75</td>
<td>1.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>16</td>
<td>329.35</td>
<td>1697.98</td>
<td>16</td>
<td>373.06</td>
<td>1314.90</td>
<td>0.83</td>
<td>1.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>5</td>
<td>602.92</td>
<td>800.10</td>
<td>5</td>
<td>810.61</td>
<td>1629.28</td>
<td>1.07</td>
<td>2.28</td>
</tr>
<tr>
<td>$4$ 2 intermediate</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
<td>550.60</td>
<td>851.57</td>
<td>9</td>
<td>811.42</td>
<td>1660.00</td>
<td>1.17</td>
<td>1.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 4.7: Average optimality gap for instances with $m = 2, d = 2$.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>AP models</th>
<th>Average optimality gap at time $t$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>$m$</td>
<td>$d$</td>
<td>$Q_{\text{max}}$</td>
</tr>
<tr>
<td>8</td>
<td>low</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>2</td>
<td>intermediate</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>high</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
</tbody>
</table>
Table 4.8: Average optimality gap for instances with $m = 2, d = 5$.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>AP models</th>
<th>Average optimality gap at time $t$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>$d$</td>
<td>$Q_{\text{max}}$</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>low</td>
<td>8</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>intermediate</td>
<td>8</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>high</td>
<td>8</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
</tbody>
</table>

NA: for these instances, the root LP relaxations of the formulations did not solve in less than 5s.
Table 4.9: Average optimality gap for instances with $m = 4, d = 2$.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>AP models</th>
<th>$n$</th>
<th>Average optimality gap at time $t$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>$m$</td>
<td>$d$</td>
<td>$Q_{\text{max}}$</td>
<td>Range</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>8</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>20</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>8</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>20</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>12</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>16</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>enhanced</td>
</tr>
</tbody>
</table>

NA: for these instances, the root LP relaxations of the formulations did not solve in less than 5s.
5. TWO MACHINES, AND ZERO INTERMEDIATE STORAGE

In this chapter, we consider Problems PFSPP-DTZ and PFSPP-CTZ, in which the flow shop has two machines with zero intermediate storage. Recall that in Problem PFSPP-DTZ, we are given a discrete speed set, while in Problem PFSPP-CTZ each machine can process jobs at any speed within a continuous interval. We will discuss these two problems in Sections 5.1 and 5.2 respectively.

5.1 Discrete speeds

As we mentioned in Chapter 3, in Problem PFSPP-DTZ we have a discrete speed set $S = \{s_1, \ldots, s_d\}$ and a set of two machines $\mathcal{M} = \{1, 2\}$, and there is zero intermediate storage in the flow shop. We will show that this variant of the PFSPP problem can be transformed into an instance of the asymmetric traveling salesperson problem (TSP). Recall that in the asymmetric TSP, we are given a complete directed graph and arc distances, and the task is to find a shortest tour in the graph. This transformation is inspired by a similar transformation of the classic permutation flow shop problem with zero intermediate storage by Reddi and Ramamoorthy (1972).

Before we describe this transformation, we need to establish the notion of a “block” in a schedule for a flow shop with no intermediate storage. Consider the following example.

Example 5.1 In a two machine flow shop with zero intermediate storage, suppose jobs $r, j, k, l$ are processed successively in a schedule. For example, consider the following Gantt chart.
Note that in this example, because there is zero intermediate storage, job \( k \) cannot leave machine 1 until job \( j \) is completed on machine 2, and so job \( l \) cannot be started on machine 1 until the completion time of job \( j \). We can view any feasible schedule for Problem PFSPP-DTZ as the combination of \( n + 1 \) blocks: a block \((j, k)\) is a subsequence such that job \( j \) is processed on machine 2, and job \( k \) is processed on machine 1. We can process each block with overlap (e.g. block \((j, k)\) in the Gantt chart above), or without overlap (e.g. block \((r, j)\)). Moreover, when the permutation of the jobs is fixed, we only need to minimize the total processing time of each block in order to find an optimal schedule.

For any feasible schedule, we define the following quantities. Let \( p(j, k) \) denote the minimum total processing time of block \((j, k)\). Let \( p_1(j, k) \) be the minimum total processing time of block \((j, k)\) when it is processed without overlap, and let \( p_2(j, k) \) be the minimum total processing time of block \((j, k)\) when jobs \( j \) and \( k \) are processed with overlap while respecting the power consumption threshold. Recall that \( s_{ij}^\ast \) is the maximum speed at which job \( j \) can be processed on machine \( i \) individually without violating the power consumption threshold. Then, it is straightforward to see that the following holds for optimal schedules.

**Lemma 5.1**

(a) In an optimal schedule for Problem PFSPP-DTZ, for each block \((j, k)\) of jobs, we have that 
\[
p_1(j, k) = p_{2js_j^2} + p_{kls_k^1}.
\]
In addition, we have that 
\[
p_2(j, k) = \min\{\max\{p_{2js_j^2}, p_{kls_k^1}\} : q_{2js_j^2} + q_{kls_k^1} \leq Q_{\max}, s_{2j} \in S, s_{1k} \in S\}.
\]

(b) In an optimal schedule for Problem PFSPP-DTZ, for each block \((j, k)\) of jobs, the total processing time \( p(j, k) \) is 
\[
\min\{p_1(j, k), p_2(j, k)\}.
\]
Note that $p(j, k)$ is not necessarily equal to $p(k, j)$. Using the above lemma, we obtain the following result.

**Theorem 5.1** Problem PFSP-PP-DTZ can be viewed as an instance of the asymmetric traveling salesperson problem.

**Proof** For convenience, we introduce a new job 0 with $p_{10} = p_{20} = 0$, and define $p(0, j) = p_{1j}s_{1j}$ and $p(j, 0) = p_{2js_{2j}}^2$ for $j = 1, \ldots, n$. Denote $j_0 = j_{n+1} = 0$. Then the makespan of schedule $(j_1, j_2, \ldots, j_n)$ is equal to $\sum_{i=0}^{n} p(j_i, j_{i+1})$. We construct a complete graph $G = (V, E)$ in which $V = \{0, 1, \ldots, n\}$. We define the distance from node $j$ to $k$ as $D_{jk} = p(j, k)$ for $j, k \in \{0, 1, \ldots, n\}$ such that $j \neq k$, and $D_{jj} = +\infty$ for $j = 0, \ldots, n$. Then Problem PFSP-PP-DTZ is equivalent to finding the shortest tour in $G$ with arc distances $D$. 

5.2 Continuous speeds

In this section we will consider Problem PFSP-CTZ, in which the flow shop has two machines with zero intermediate storage, each machine can process jobs at any speed within a continuous interval, and the power consumption of a machine processing a job at speed $s$ is $s^\alpha$ for some constant $\alpha > 1$. Given $Q_{\text{max}}$ as the threshold for peak power consumption, we define $s_{\text{max}} = (Q_{\text{max}})^{\frac{1}{\alpha}}$, and let the speed set $S$ be the continuous interval $[0, s_{\text{max}}]$. Recall that $p_{ij}$ is the work required for job $j$ on machine $i$, $s_{ij} \in [0, s_{\text{max}}]$ is the chosen speed to process job $j$ on machine $i$, and $p_{ij}/s_{ij}$ is the processing time of job $j$ on machine $i$.

5.2.1 Arbitrary work volume of jobs across machines

**Lemma 5.2** In any optimal schedule for Problem PFSP-CTZ, if job $j$ immediately precedes job $k$, then $s_{1k}^\alpha + s_{2j}^\alpha = Q_{\text{max}} = s_{\text{max}}^\alpha$. Moreover, each block $(j, k)$ with $j \neq 0$ and $k \neq 0$ in an optimal schedule is processed with overlap, and $C_{2j} = C_{1k}$. 

Proof Note that at any time, the total power consumption of the two machines must be exactly $Q_{\text{max}}$; otherwise we can increase the speeds of the jobs on the two machines so that the total power consumption is $Q_{\text{max}}$, and decrease the makespan.

Consider a block $(j, k)$ with $j \neq 0$ and $k \neq 0$ in an optimal schedule. If block $(j, k)$ is processed without overlap in the optimal schedule, then job $j$ and $k$ must be processed at the maximum speed $s_{\text{max}}$. That is, the minimum total processing time for block $(j, k)$ without overlap is $p_1(j, k) = (p_{2j} + p_{1k})/(Q_{\text{max}})^{\frac{1}{\alpha}}$.

If block $(j, k)$ is processed with overlap in the optimal schedule, then it must be nondelay; otherwise we can process the delayed job (job $k$) earlier with the same speed as follows:

\[
\begin{array}{c}
\text{machine 1} & \cdots & j & \cdots & k & \cdots \\
\text{machine 2} & \cdots & \text{ } j \text{ } & \text{ } k \text{ } & \cdots \\
\end{array}
\]

This way, we can decrease the makespan without violating the power consumption constraints. As a result, when block $(j, k)$ is processed with overlap, then job $j$ on machine 2 and job $k$ on machine 1 must be processed at the same start time. Moreover, we also have $C_{2j} = C_{1k}$. Otherwise, without loss of generality, suppose that $C_{2j} > C_{1k}$. Then we can decrease the speed of job $k$ and increase the speed of job $j$ until $C_{2j} = C_{1k}$.

\[
\begin{array}{c}
\text{machine 1} & \boxed{k} & \boxed{j} & \rightarrow & \boxed{k} & \boxed{j} \\
\text{machine 2} & \boxed{j} & \boxed{k} & \rightarrow & \boxed{k} & \boxed{j} \\
\end{array}
\]

This way, the total processing time of block $(j, k)$ decreases, and so the makespan also decreases. Therefore for any block $(j, k)$ with overlap, we have $p_{2j}/s_{2j} = p_{1k}/s_{1k} = p_{2}(j, k)$. Because $s_{2j}^{\alpha} + s_{1k}^{\alpha} = Q_{\text{max}}$, we obtain that $p_{2}(j, k) = (p_{1k}^{\alpha} + p_{2j}^{\alpha})^{\frac{1}{\alpha}}/(Q_{\text{max}})^{\frac{1}{\alpha}}$. Since $p_1(j, k) > p_2(j, k)$ for any $\alpha > 1$, we have $p(j, k) = p_2(j, k)$.

Define $D_{jk} = (p_{1k}^{\alpha} + p_{2j}^{\alpha})^{\frac{1}{\alpha}}$ for $j, k \in \{0, \ldots, n\}$ such that $j \neq k$. Since $p(j, k) = D_{jk}/(Q_{\text{max}})^{\frac{1}{\alpha}}$ for all $j, k \in \{0, \ldots, n\}$ such that $j \neq k$, our problem is equivalent to
finding a permutation \((j_1, \ldots, j_n)\) of the jobs that minimizes \(\sum_{i=0}^{n} D_{j_i, j_{i+1}}\). Similar to the proof in Theorem 5.1, if we interpret \(D_{jk}\) as the distance of the arc from node \(j\) to node \(k\) in a complete directed graph on \(\{0, 1, \ldots, n\}\), and define \(D_{jj} = +\infty\) for \(j = 0, \ldots, n\), then this variant of the PFSP problem is also a special case of the asymmetric TSP.

**Theorem 5.2** Problem PFSP-CTZ can be viewed as an instance of the asymmetric traveling salesperson problem.

### 5.2.2 Consistent work volume of jobs across machines

Although the asymmetric TSP is an NP-hard problem, many of its special cases can be solved efficiently in polynomial time. One such special case is when the arc distances satisfy the so-called *Demidenko conditions*, which state that the matrix \(D \in \mathbb{R}^{(n+1) \times (n+1)}\) of arc distances satisfies the following conditions: for all \(i, j, l \in \{0, 1, \ldots, n\}\) such that \(i < j < j + 1 < l\), we have

\[
D_{ij} + D_{j,j+1} + D_{j+1,l} \leq D_{i,j+1} + D_{j+1,j} + D_{jl}, \tag{5.1}
\]

\[
D_{l,j+1} + D_{j+1,j} + D_{ji} \leq D_{lj} + D_{j,j+1} + D_{j+1,i}, \tag{5.2}
\]

\[
D_{ij} + D_{l,j+1} \leq D_{lj} + D_{i,j+1}, \tag{5.3}
\]

\[
D_{ji} + D_{j+1,l} \leq D_{jl} + D_{j+1,i}. \tag{5.4}
\]

We say that a tour on cities \(0, 1, \ldots, n\) is *pyramidal* if it is of the form \((0, i_1, \ldots, i_r, j_1, \ldots, j_{n-r-1})\), where \(i_1 < i_2 < \cdots < i_r\) and \(j_1 > j_2 > \cdots > j_{n-r-1}\).

Demidenko (1979) showed the following for the asymmetric TSP.

**Theorem 5.3 (Demidenko 1979)** If \(D \in \mathbb{R}^{(n+1) \times (n+1)}\) satisfies the Demidenko conditions, then for any tour there exists a pyramidal tour of no greater cost. Moreover, a minimum cost pyramidal tour can be determined in \(O(n^2)\) time.
Coming back to Problem PFSSP-CTZ, suppose the work volume of jobs is consistent across machines: that is, for any two jobs \( j, k \in J \), we have that \( p_{1j} \leq p_{1k} \) implies \( p_{2j} \leq p_{2k} \). Then we have the following theorem.

**Theorem 5.4** If the work required is consistent across machines, then there exists an optimal schedule for Problem PFSSP-CTZ that corresponds to a pyramidal TSP tour, and such a schedule can be found in \( O(n^2) \) time.

**Proof** Fix \( i, j, l \in \{0, 1, \ldots, n\} \) such that \( i < j < j + 1 < l \). Without loss of generality, suppose \( p_{11} \leq p_{12} \leq \cdots \leq p_{1n} \) and \( p_{21} \leq p_{22} \leq \cdots \leq p_{2n} \). We can do this since the work is assumed to be consistent across machines. Therefore, \( p_{1i} \leq p_{1j} \leq p_{1,j+1} \leq p_{1l} \) and \( p_{2i} \leq p_{2j} \leq p_{2,j+1} \leq p_{2l} \). We prove that \( D_{jk} = (p_{1k}^\alpha + p_{ij}^\alpha)^{1/\alpha} \) for \( j, k \in \{0, 1, \ldots, n\} \) such that \( j \neq k \) and \( D_{jj} = +\infty \) for \( j = 0, 1, \ldots, n \) satisfies the Demidenko conditions.

Conditions (5.1): Let \( g(x) = (x^\alpha + p_{2i}^\alpha)^{1/\alpha} - (x^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} \). Then it is straightforward to verify that \( g'(x) \geq 0 \), and so we have \( g(p_{1,j+1}) \geq g(p_{1j}) \), i.e.

\[
(p_{1,j+1}^\alpha + p_{2i}^\alpha)^{1/\alpha} - (p_{1j+1}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} \geq (p_{1j}^\alpha + p_{2i}^\alpha)^{1/\alpha} - (p_{1j}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha}.
\]

This is equivalent to

\[
(p_{1,j+1}^\alpha + p_{2i}^\alpha)^{1/\alpha} - (p_{1j}^\alpha + p_{2i}^\alpha)^{1/\alpha} + (p_{1j}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} - (p_{1,j+1}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} \geq (p_{1,j+1}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} - (p_{1,j+1}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha}.
\]

Let \( f(x) = (x^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} - (x^\alpha + p_{2j}^\alpha)^{1/\alpha} \). Similarly we can prove that \( f'(x) \leq 0 \), and so

\[
(p_{1,j+1}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} - (p_{1j+1}^\alpha + p_{2j}^\alpha)^{1/\alpha} \geq (p_{1j}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} - (p_{1j}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha}.
\]

So we have

\[
(p_{1,j+1}^\alpha + p_{2i}^\alpha)^{1/\alpha} - (p_{1j}^\alpha + p_{2i}^\alpha)^{1/\alpha} + (p_{1j}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} - (p_{1,j+1}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} \geq (p_{1j}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha} - (p_{1j}^\alpha + p_{2,j+1}^\alpha)^{1/\alpha}.
\]
or equivalently

\[ D_{i,j+1} - D_{ij} + D_{j+1,j} - D_{j,j+1} \geq D_{j+1,l} - D_{jl}, \]

which indicates that conditions (5.1) are satisfied.

Conditions (5.2): Similar to the proof of conditions (5.1).

Conditions (5.3): Let \( h(x) = (x^\alpha + p_2^\alpha)^{\frac{1}{\alpha}} - (x^\alpha + p_2^\alpha)^{\frac{1}{\alpha}}. \) Then it is straightforward to verify that \( h'(x) \geq 0, \) so we have

\[ (p_{i,j+1}^\alpha + p_{2i}^\alpha)^{\frac{1}{\alpha}} - (p_{1,i,j+1}^\alpha + p_{2i}^\alpha)^{\frac{1}{\alpha}} \geq (p_{1,j}^\alpha + p_{2i}^\alpha)^{\frac{1}{\alpha}} - (p_{1,j}^\alpha + p_{2i}^\alpha)^{\frac{1}{\alpha}}, \]

which indicates that conditions (5.3) are satisfied.

Conditions (5.4): Similar to the proof of conditions (5.3).

In general, it may be the case that different jobs have their own speed ranges and power functions (e.g. Bansal et al. 2009). In other words, it may be the case that each job \( j \) has a power function of the form \( a_j s_j^\alpha + c_j, \) where \( s_j \in [s_j^{\min}, s_j^{\max}] \triangleq S_j. \) Under this environment, we may obtain different functions \( p(j,k) \) with respect to \( p_{1k} \) and \( p_{2j} \) for each block \((j,k)\) in Problem PFSP-CTZ. Using the Demidenko conditions, we can extend Theorem 5.4 as follows.

**Theorem 5.5** For Problem PFSP-CTZ, if the functions \( p(j,k) \) for all \( j,k \in \{0,1,\ldots,n\} \) with \( j \neq k \) are determined by a twice differentiable function \( g(x,y) \) such that \( p(j,k) = g(p_{1k},p_{2j}) \) and \( \partial^2 g/\partial x \partial y < 0, \) then there must exist an optimal schedule that corresponds to a pyramidal tour.

**Proof** Fix \( i,j,l \in \{0,1,\ldots,n\} \) such that \( i < j < j+1 < l. \) Without loss of generality, suppose \( p_{11} \leq p_{12} \leq \cdots \leq p_{1n} \) and \( p_{21} \leq p_{22} \leq \cdots \leq p_{2n}. \) Similar to the proof of Theorem 5.4, we show that the matrix \( D \) such that \( D_{jk} = p(j,k) \) for all \( j,k \in \{0,1,\ldots,n\} \) such that \( j \neq k \) satisfies the Demidenko conditions under the above assumption.
To show conditions (5.1) are satisfied, we need to prove that

\[ g(p_{1,j+1}, p_{2i}) + g(p_{1j}, p_{2,j+1}) + g(p_{1j}, p_{2j}) \geq g(p_{1j}, p_{2i}) + g(p_{1j+1}, p_{2j}) + g(p_{1i}, p_{2,j+1}). \]

Let \( h(x) = g(x, p_{2i}) - g(x, p_{2,j+1}) \). Then

\[ \frac{\partial h}{\partial x} = \frac{\partial g(x, p_{2i})}{\partial x} - \frac{\partial g(x, p_{2,j+1})}{\partial x}. \]

Because \( \partial^2 g/\partial x \partial y < 0 \), we obtain that \( \partial h/\partial x \geq 0 \). So

\[ g(p_{1,j+1}, p_{2i}) - g(p_{1,j+1}, p_{2,j+1}) \geq g(p_{1j}, p_{2i}) - g(p_{1j}, p_{2,j+1}). \]

Similarly, we can also prove that

\[ g(p_{1,j+1}, p_{2,j+1}) - g(p_{1,j+1}, p_{2j}) \geq g(p_{1i}, p_{2,j+1}) - g(p_{1i}, p_{2j}). \]

Combining the above two results, conditions (5.1) are satisfied.

Using the same arguments as in the proof of Theorem 5.4 and above, we can verify conditions (5.2), (5.3), and (5.4) similarly.

**5.2.3 Equal work volume of jobs across machines**

If the work required for each job is equal on each machine – that is, for any job \( j \in J \), we have that \( p_{1j} = p_{2j} = p_j \) – then we can further refine the results of the previous subsection. By Theorem 5.4, there exists an optimal pyramidal tour for this variant of Problem PFSPP-CTZ. For this variant, we claim that there must exist an optimal schedule of the form \((1, 3, 5, \ldots, n, \ldots, 6, 4, 2)\), assuming that \( p_1 \leq p_2 \leq \cdots \leq p_n \).

**Lemma 5.3** Consider a subsequence of an optimal schedule as follows:

<table>
<thead>
<tr>
<th>machine 1</th>
<th>( \cdots )</th>
<th>( i )</th>
<th>( j )</th>
<th>( k )</th>
<th>( \cdots )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( \cdots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>machine 2</td>
<td>( \cdots )</td>
<td>( i )</td>
<td>( j )</td>
<td>( k )</td>
<td>( \cdots )</td>
<td>( a )</td>
<td>( b )</td>
<td>( c )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>
If \( p_i \leq p_c \), then we must have \( p_j \leq p_b \).

**Proof** By contradiction. Suppose in an optimal schedule \( \sigma_1 \), we have \( p_i \leq p_c \) but \( p_j > p_b \). Consider rescheduling the jobs between job \( i \) and \( c \) in reverse order as follows (i.e. \( i \rightarrow b \rightarrow a \rightarrow \cdots \rightarrow k \rightarrow j \rightarrow c \)). Denote this new schedule as \( \sigma_2 \).

Denote the makespan of schedule \( \sigma_i \) as \( C_{\sigma_i}, i = 1, 2 \). Then we have

\[
C_{\sigma_1} - C_{\sigma_2} = \frac{(p_i^c + p_j^c)^{\frac{1}{a}} + (p_c^a + p_b^a)^{\frac{1}{a}} - (p_i^a + p_b^a)^{\frac{1}{a}} - (p_c^a + p_j^a)^{\frac{1}{a}}}{(Q_{\text{max}})^{\frac{1}{a}}}.
\]

Similar to the proof of Theorem 5.4, we can show that \( C_{\sigma_1} - C_{\sigma_2} > 0 \), which contradicts schedule \( \sigma_1 \) being optimal.

**Lemma 5.4** For any optimal schedule, suppose that the first job processed is job \( i \), the second is \( b \) and the last is \( c \). Without loss of generality, if we assume that \( p_i \leq p_c \), then there must exist an optimal schedule which satisfies \( p_b \geq p_c \).

**Proof** By contradiction. If an optimal schedule \( \sigma_1 \) does not satisfy \( p_b \geq p_c \), i.e. \( p_b < p_c \), then we reschedule job \( i \) so that \( i \) becomes the last job in the schedule, while maintaining the ordering of all the other jobs. We denote the new schedule as \( \sigma_2 \):

Similar to the proof of Theorem 5.4, it is easy to verify that

\[
C_{\sigma_1} - C_{\sigma_2} = \frac{p_c + (p_i^a + p_b^a)^{\frac{1}{a}} - p_b - (p_i^a + p_c^a)^{\frac{1}{a}}}{(Q_{\text{max}})^{\frac{1}{a}}} > 0,
\]

and so schedule \( \sigma_1 \) is not optimal.
Theorem 5.6 Assuming that $p_1 \leq p_2 \leq \cdots \leq p_n$, there must exist an optimal schedule of the form $(1,3,5,\ldots,n,\ldots,6,4,2)$.

Proof For simplicity, we denote the workload of the job in $j$th position of an optimal schedule as $p_{\sigma(j)}$. Without loss of generality, we assume that $p_{\sigma(1)} \leq p_{\sigma(n)}$. By Lemma 5.3, we obtain that $p_{\sigma(i)} \leq p_{\sigma(n-i+1)}$ for $i = 1,\ldots,[n/2]$. By Lemma 5.4, we have $p_{\sigma(2)} \geq p_{\sigma(n)}$. Consider the subsequence of an optimal schedule from the second position to the $n$th position. By Lemma 5.3, we obtain that $p_{\sigma(i+1)} \geq p_{\sigma(n-i+1)}$ for $i = 1,\ldots,[n/2]$. Combining the above results, we have $p_{\sigma(1)} \leq p_{\sigma(n)} \leq p_{\sigma(2)} \leq p_{\sigma(n-1)} \leq p_{\sigma(3)} \leq \cdots$, and so there exists an optimal schedule of the form $(1,3,5,\ldots,n,\ldots,6,4,2)$. 

\vspace{1cm}
Part III
Scheduling on a single machine under time-of-use tariffs
6. THE UNIFORM-SPEED MACHINE CASE

In this chapter, we consider the SMSEC problem with uniform-speed machine. We first prove that Problem SMSEC-U is strongly NP-hard and inapproximable within a constant factor unless \( P = NP \), which reveals some of the intrinsic difficulties of the SMSEC problem. Then, we investigate the structural properties of an optimal schedule for Problem SMSEC-U-pmtn. Finally, we propose an exact polynomial-time algorithm for Problem SMSEC-U-pyr.

6.1 NP-hardness and inapproximability of Problem SMSEC-U

Theorem 6.1 Problem SMSEC-U is strongly NP-hard, and in fact inapproximable within a constant factor, unless \( P = NP \).

Proof The proof is based on a reduction from 3-PARTITION to Problem SMSEC-U. The 3-PARTITION problem is described as follows: Given a set \( S = \{1, 2, \ldots, 3m\} \) and positive integers \( a_1, a_2, \ldots, a_{3m}, b \) such that \( b/4 < a_j < b/2 \) for all \( j \in S \) and \( \sum_{j \in S} a_j = mb \), does there exist a partition of \( S \) with \( m \) 3-element subsets \( S_i \) such that \( \sum_{j \in S_i} a_j = b \) for all \( i = 1, 2, \ldots, m \)?

Given any instance \( \mathcal{I}_1 \) of 3-PARTITION, we construct an instance \( \mathcal{I}_2 \) of Problem SMSEC-U as follows. There are \( m \) periods of duration \( b \) and \( m - 1 \) periods of duration 1. The TOU tariff is shown in Figure 6.1: the electricity price of period \( [lb + l - 1, lb + l) \) is \( c_l = 1 \) for \( l = 1, 2, \ldots, m - 1 \), and the electricity price of period \( [kb + k, (k + 1)b + k) \) is \( c_k = 0 \) for \( k = 0, 1, \ldots, m - 1 \).
The number of jobs $n$ is equal to $4m - 1$. The processing time and power consumption for each job $j$ is as follows:

\[
p_j = 1, \quad q_j = 0 \quad \text{for } j = 1, 2, \ldots, m - 1; \\
p_j = a_{j-m+1}, \quad q_j = 1 \quad \text{for } j = m, m + 1, \ldots, 4m - 1.
\]

Then a schedule with $E = 0$ exists if and only if every job $j \in \{1, 2, \ldots, m - 1\}$ can be processed between time periods $[lb + l - 1, lb + l)$ for $l = 1, 2, \ldots, m - 1$. This can be done if and only if the remaining jobs can be partitioned over the $m$ intervals of length $b$, which can be done if and only if the 3-PARTITION instance has a solution. Since 3-PARTITION is strongly NP-hard (Garey and Johnson 1979), we conclude that Problem SMSEC-U is strongly NP-hard.

Now fix some $\rho > 1$, and assume that there exists a $\rho$-approximation algorithm for Problem SMSEC-U. We will prove that the output of the algorithm to instance $I_2$ allows us to conclude whether there is a solution to any given instance $I_1$ of 3-PARTITION in polynomial time.

We use the $\rho$-approximation algorithm to solve $I_2$. The solution of the algorithm has cost $E_{\text{algo}}$ which satisfies $E_{\text{opt}} \leq E_{\text{algo}} \leq \rho E_{\text{opt}}$, where $E_{\text{opt}}$ is the optimal cost. We consider the following two cases.
1. $E_{\text{algo}} = 0$. In this case, $E_{\text{opt}} = 0$; that is, every job $j \in \{1, 2, \ldots, m - 1\}$ are processed between periods $[lb + l - 1, lb + l)$ for $l = 1, 2, \ldots, m - 1$, and the remaining jobs are partitioned over the $m$ intervals of length $b$, and therefore such partition for jobs $m, m + 1, \ldots, 4m - 1$ is a solution to $\mathcal{I}_1$ of 3-PARTITION.

2. $E_{\text{algo}} > 0$. In this case, there is no solution to $\mathcal{I}_1$ of 3-PARTITION. Otherwise, if there is a solution to $\mathcal{I}_1$, that is, we can find a partition of $\mathcal{S}$ with $m$ 3-element subsets $S_i$ such that $\sum_{j \in S_i} a_j = b$ for all $i = 1, \ldots, m$. Then we can process the jobs $m, m + 1, \ldots, 4m - 1$ in period $[kb + k, (k + 1)b + k)$ for $k = 0, 1, \ldots, m - 1$, according to the elements in each $S_i$, and the other jobs in period $[lb + l - 1, lb + l)$ for $l = 1, 2, \ldots, m - 1$, and the corresponding electricity cost is 0, i.e. $E_{\text{opt}} = 0$.

Since $E_{\text{algo}} \leq \rho E_{\text{opt}}$, we have $E_{\text{algo}} = 0$, which is a contradiction.

Therefore, we can conclude in polynomial time whether there is a solution to the instance $\mathcal{I}_1$ of 3-PARTITION from the output of the algorithm for $\mathcal{I}_2$. Since 3-PARTITION is strongly NP-hard (Garey and Johnson 1979), Problem SMSEC-U is inapproximable within a constant factor, unless $P = NP$.

6.2 Properties of optimal schedules for Problem SMSEC-U-pmtn

For simplicity, we denote the remaining idle time in period $k$ while we schedule jobs as $D_k$ for $k \in \mathcal{P}$. Initially $D_k = d_k$ for each period $k \in \mathcal{P}$. The following lemma describes some structural properties of optimal schedules for Problem SMSEC-U-pmtn.

**Lemma 6.1** For any optimal schedule $\sigma_1$ of Problem SMSEC-U-pmtn, if one unit of job $i$ was processed in period $h$, one unit of job $j$ was processed in period $k$, and $c_h < c_k$, then $q_i \geq q_j$.

**Proof** By contradiction. If schedule $\sigma_1$ does not satisfy $q_i \geq q_j$, i.e. $q_i < q_j$, then we perform a pairwise interchange on the two units of jobs $i$ and $j$, while maintaining the schedule of all the other units. Denote this new schedule as $\sigma_2$; see Figure 6.2.
Denote the total electricity cost for the two units of job $i$ and $j$ in schedule $\sigma_1$ and $\sigma_2$ as $E_1$ and $E_2$, respectively. Then $E_1 = q_ic_h + q_jc_k$ and $E_2 = q_jc_h + q_ic_k$. Therefore, $E_1 - E_2 = (q_i - q_j)(c_h - c_k)$. Since $c_h < c_k$ and $q_i < q_j$, we obtain that $E_1 > E_2$, which contradicts schedule $\sigma_1$ being optimal.

The above lemma implies that given an optimal preemptive schedule, the higher the power consumption of a job, the lower the electricity price of the period it is assigned. Therefore, we can construct an optimal preemptive schedule as follows.

Algorithm 6.2.1 Exact polynomial-time algorithm for Problem SMSEC-U-pmtn

Require: $p_j, q_j$ for $j \in J$, $c_k, d_k$ for $k \in P$, jobs can be processed preemptively.

1: Sort the periods according to permutation $\phi$ such that $c_{\phi(1)} \leq c_{\phi(2)} \leq \cdots \leq c_{\phi(K)}$.
2: Sort the jobs such that $q_1 \geq q_2 \cdots \geq q_n$.
3: for $j = 1$ to $n$ do
4:     for $k = 1$ to $K$ do
5:         if $D_{\phi(k)} > 0, p_j > 0$ then
6:             Calculate $t = \min\{D_{\phi(k)}, p_j\}$.
7:             Schedule $t$ units of job $j$ into period $\phi(k)$.
8:             Update $p_j = p_j - t, D_{\phi(k)} = D_{\phi(k)} - t$.
9:         end if
10:    end for
11: end for

Theorem 6.2 Algorithm 6.2.1 constructs an optimal schedule for Problem SMSEC-U-pmtn in polynomial time.
6.3 An exact polynomial-time algorithm for Problem SMSEC-U-pyr

In this section we consider Problem SMSEC-U-pyr, in which all the jobs have the same work volume, and the electricity prices are pyramidal; that is, the electricity prices satisfy $c_1 < c_2 < \cdots < c_{h-1} < c_h > c_{h+1} > \cdots > c_K$. Figure 6.3 gives an example of pyramidal TOU tariffs.

![Figure 6.3: An example of pyramidal TOU tariffs.](image)

It is easy to check if a given set of jobs can be scheduled within the given time horizon. In this chapter and the next we always assume that a feasible solution exists. Before describing our exact polynomial-time algorithm for Problem SMSEC-U-pyr in detail, let us consider a small example and see what its optimal schedule looks like.

**Example 6.1** Table 6.1(a) shows a six-period pyramidal TOU tariff with their durations and electricity prices. Table 6.1(b) gives six jobs with their processing time and power consumption. Figure 6.4 illustrates an optimal schedule for the given instance. For simplicity, we define the forward direction as the direction in which time increases, and the backward direction as the direction in which time decreases. Through some examination, we find that there always exists an optimal schedule as follows: for the jobs processed before (after) the period with highest electricity price, i.e., period 3 in this example, they are processed continuously in the forward (backward) direction starting from time 0 ($T$) in order of decreasing power consumption.

Inspired by this observation, we propose an algorithm that schedules jobs simultaneously in the forward direction starting at time 0 and in the backward direction...
Table 6.1: An instance with pyramidal TOU tariffs.

(a) A pyramidal TOU tariff

<table>
<thead>
<tr>
<th>Period ($k$)</th>
<th>Duration ($d_k$)</th>
<th>Price ($c_k$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

(b) Jobs for this instance

<table>
<thead>
<tr>
<th>Job ($j$)</th>
<th>Processing time ($p_j$)</th>
<th>Power ($q_j$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 6.4.: An optimal schedule for the instance given in Example 6.1.

starting at time $T$. Let $t_f$ and $t_b$ be the earliest idle time in the forward and backward direction respectively. In other words, $t_f$ and $T - t_b$ are the total processing time of jobs before processing the next job in the forward and backward direction.
respectively. Initially we set $t_f = 0$, and $t_b = T$. Note that we assume that a feasible solution exists—i.e., $t_f + (T - t_b) \leq T$—so we always have $t_f \leq t_b$. Since all the jobs have the same processing time, we define the total price at time $t_f$ ($t_b$) as the total electricity price (not cost) for processing a job starting at $t_f$ ($t_b$) in the forward (backward) direction. For simplicity, we denote $A(t_f)$ ($\tilde{A}(t_b)$) as the total price at time $t_f$ ($t_b$) in the forward (backward) direction. Note that $A(t_f)$ and $\tilde{A}(t_b)$ are job-independent, since $p_j = p$ for all $j \in J$. In Example 6.1, after we schedule jobs 1 and 2, $t_f = 0 + p_1 = 2$, $t_b = T - p_2 = \sum_{k=1}^{6} d_k - p_2 = 14 - 2 = 12$. Job 3 is processed starting at time $t_b = 12$ in the backward direction, with one unit of time in period 6 and one unit of time in period 5, so the total price at $t_b = 12$ in the backward direction is $\tilde{A}(12) = (1 \times c_5) + (1 \times c_6) = 5$.

In what follows, we show that Algorithm 6.3.1 (below) constructs an optimal schedule for Problem SMSEC-U-pyr. Let $J_f$ and $J_b$ be the set of jobs that are processed in the forward and backward direction respectively according to Algorithm 6.3.1. Suppose job $l$ ($r$) is the last job processed in $J_f$ ($J_b$). In addition, we denote the time when we start processing job $j$ in $J_f$ ($J_b$) in the forward (backward) direction as $S_j$, and the time when we finish processing job $j$ in $J_f$ ($J_b$) in the forward (backward) direction as $C_j$; see Figure 6.5.

**Algorithm 6.3.1** Exact polynomial-time algorithm for Problem SMSEC-U-pyr

**Require:** $p_j = p, q_j$ for $j \in J$, a pyramidal TOU tariff $c_k$, and $d_k$ for $k \in P$.

1. Sort the jobs such that $q_1 \geq q_2 \geq \cdots \geq q_n$.
2. Initialize $t_f = 0, t_b = T$.
3. for $j = 1$ to $n$ do
   4. Calculate $A(t_f)$ and $\tilde{A}(t_b)$.
   5. if $A(t_f) < \tilde{A}(t_b)$ then
      6. Schedule job $j$ in the forward direction starting at $t_f$.
      7. Update $t_f = t_f + p$.
   8. else
      9. Schedule job $j$ in the backward direction starting at $t_b$.
     10. Update $t_b = t_b - p$.
  11. end if
4. end for
Figure 6.5.: A schedule generated by Algorithm 6.3.1.

**Lemma 6.2** For any schedule generated by Algorithm 6.3.1, the total price at time $S_j$ for $j \in J_f$ ($j \in J_b$) is monotonically increasing in the forward (backward) direction.

**Proof** Without loss of generality, we consider the properties of total prices at the start time of jobs in the forward direction.

Case 1: $C_l \leq \sum_{k=1}^h d_k$; that is, job $l$ completes before the end of period $h$. Since $c_1 < c_2 \cdots < c_h$, the lemma follows directly.

Case 2: $C_l > \sum_{k=1}^h d_k$. First, we show that we must have $S_l \leq \sum_{k=1}^{h-1} d_k$. Otherwise, suppose the earliest idle time for processing job $l$ in the forward and backward direction are $t_{fl}$ and $t_{bl}$, respectively. Since there always exists a feasible solution, we have $t_{fl} + p \leq t_{bl}$. If $S_l = t_{fl} > \sum_{k=1}^{h-1} d_k$, since $c_h > c_{h+1} > \cdots > c_K$, we have $A(t_{fl}) = \tilde{A}(T - (t_{fl} + p)) \geq \tilde{A}(t_{bl})$. As a result, according to Algorithm 6.3.1, job $l$ must be processed in the backward direction, which contradicts job $l \in J_f$.

Now suppose job $l'$ is processed immediately before job $l$. Since $C_{l'} = S_l \leq \sum_{k=1}^{h-1} d_k$, we obtain that $A(S_i) \leq A(S_j)$ for $1 \leq i < j \leq l'$. Now we will prove that $A(S_{l'}) \leq A(S_l)$. Suppose the earliest idle time in the forward and backward direction are $t_{fl'}$ and $t_{bl'}$ respectively right before we start to process job $l'$ in Algorithm 6.3.1. Since $l' \in J_f$, we know that $A(S_{l'}) = A(t_{fl'}) < \tilde{A}(t_{bl'})$. Note that job $l$ is processed within
and the total price at any time in this interval is always greater than
$A(t_{fv})$. As a result, we have $A(S_{f}) \leq A(S_{i})$.

**Theorem 6.3** Algorithm 6.3.1 is an exact polynomial-time algorithm for Problem SMSEC-U-pyr.

**Proof** For any schedule generated by Algorithm 6.3.1, using the property in
Lemma 6.2, we can easily prove that for any two jobs $j, k \in J_{f}$ (or $J_{b}$) it is impossible to reduce their electricity cost by performing a pairwise interchange on them. As a result, for any two jobs in such a schedule, we only need to prove that it is also impossible to reduce the electricity cost by performing a pairwise interchange on them when one is in $J_{f}$ and the other is in $J_{b}$.

Suppose jobs 1 to $j-1$ were already scheduled in Algorithm 6.3.1, and the current earliest idle time in the forward and backward direction is $t_{f}$ and $t_{b}$, respectively. Without loss of generality, we assume that job $j$ is processed in the forward direction; that is, $A(t_{f}) < \tilde{A}(t_{b})$.

![Diagram](image)

Now suppose all of the jobs have been processed according to Algorithm 6.3.1. We prove that the total electricity cost cannot be reduced by performing a pairwise interchange on job $j$ and any other job in $J_{b}$.

Suppose job $k$ is processed in the backward direction starting at $t_{b}$, and job $k'$ ($k''$) is any job that was processed in the backward direction earlier (later) than $t_{b}$. Denote the start time of jobs $k'$ and $k''$ in the backward direction as $t'_b$ and $t''_b$ respectively. Then according to Algorithm 6.3.1 and Lemma 6.2, we have $\tilde{A}(t'_b) \leq A(t_{f}) < \tilde{A}(t_{b}) \leq \tilde{A}(t''_b)$. Denote $E^{1}_{jk}$ as the total electricity cost of jobs $j$ and $k$ in the schedule output by Algorithm 6.3.1, and $E^{2}_{jk}$ the total electricity cost of jobs $j$ and $k$ in the schedule.
obtained by performing a pairwise interchange on them, while keeping all the other jobs in the same positions. Then we have the following.

1. \( E_{jk}^1 \leq E_{jk}^2 \). According to Algorithm 6.3.1, we know that \( q_j \geq q_k \). Since \( E_{jk}^1 = A(t_f)q_j + \tilde{A}(t_b)q_k \) and \( E_{jk}^2 = A(t_f)q_k + \tilde{A}(t_b)q_j \), we obtain that \( E_{jk}^1 - E_{jk}^2 = (A(t_f) - \tilde{A}(t_b))(q_j - q_k) \leq 0 \); that is, we cannot reduce the electricity cost by a pairwise interchange on jobs \( j \) and \( k \).

2. \( E_{jk'}^1 \leq E_{jk'}^2 \). Since job \( k' \) is processed in the backward direction before \( t_b \), we know that job \( k' \) is processed earlier than job \( j \) in Algorithm 6.3.1, and so we have \( q_{k'} \geq q_j \). Similarly we have \( E_{jk'}^1 - E_{jk'}^2 = (A(t_f) - \tilde{A}(t'_b))(q_j - q_{k'}) \leq 0 \); that is, we cannot reduce the electricity cost by a pairwise interchange on jobs \( j \) and \( k' \).

3. \( E_{jk''}^1 \leq E_{jk''}^2 \). Since job \( k'' \) is processed in the backward direction later than \( t_b \), we know that \( q_j \geq q_{k''} \). Similarly we have \( E_{jk''}^1 - E_{jk''}^2 = (A(t_f) - \tilde{A}(t''_b))(q_j - q_{k''}) \leq 0 \); that is, we cannot reduce the electricity cost by a pairwise interchange on jobs \( j \) and \( k'' \).

Using the same arguments as above, for any job \( k \in J_b \), we cannot reduce the total electricity cost by performing a pairwise interchange on job \( k \) and any other job in \( J_f \).
7. THE CONTINUOUS SPEED-SCALABLE MACHINE CASE

In this chapter, we assume that jobs can be processed at an arbitrary speed. As we mentioned earlier, if job $j$ has a work volume of $w_j$, when it is processed at speed $s$, its processing time is $w_j/s$, and its power consumption is $s^\alpha$ ($\alpha \geq 1$). We first investigate the structure of optimal preemptive schedules on a speed-scalable machine for Problem SMSEC-S-pmttn. Then we prove that Problem SMSEC-S is strongly NP-hard by exploiting this structure. In addition, we show that there is no PTAS for Problem SMSEC-S. We also investigate the structural properties of optimal schedules for special cases of Problem SMSEC-S. Finally, we propose and analyze approximation algorithms for Problem SMSEC-S by transforming preemptive schedules into non-preemptive ones.

7.1 Properties of optimal schedules for Problem SMSEC-S-pmttn

As we mentioned before, in Problem SMSEC-S-pmttn we assume that jobs can be processed preemptively. Specifically, when a job is partitioned into several parts in an optimal preemptive schedule, then the speeds for processing each part can be different. For each period $l \in \mathcal{P}$, we denote the work volume of job $j$ assigned to it as $w_{jl}$, and the total work volume assigned to period $l$ as $v_l$, i.e., $v_l = \sum_{j \in \mathcal{J}} w_{jl}$.

The following lemma describes some structural properties of an optimal preemptive schedule in a given period.

Lemma 7.1 In an optimal schedule $\sigma_1$ for Problem SMSEC-S-pmttn, suppose that period $l$ processes a total work volume of $v_l$. Then

1. the total processing time of jobs in period $l$ must be $d_l$, and
(2) each job in period \( l \) is processed at the same speed of \( v_l/d_l \).

**Proof**  
(1) By contradiction. Suppose that in schedule \( \sigma_1 \), the total processing time of jobs in period \( l \) is less than \( d_l \); that is, period \( l \) has remaining idle time \( D_l > 0 \). Then we can choose any of the jobs (e.g. job \( j \)) in period \( l \) and slow down its speed until \( D_l = 0 \) while keeping the speeds of all the other jobs the same. We denote the new schedule as \( \sigma_2 \) and the corresponding speed for job \( j \) as \( s'_j \). Then the electricity cost for job \( j \) in period \( l \) becomes \( c_l w_jl(s'_j)^{\alpha-1} \). Since \( s'_j < s_j \), this contradicts the optimality of schedule \( \sigma_1 \).

(2) Suppose jobs \( j_1, \ldots, j_k \) are processed in period \( l \) in schedule \( \sigma_1 \), with work volume of \( w_{j_1 l} > 0 \) for \( i = 1, \ldots, k \). Without loss of generality, we assume that jobs are processed in the order \( j_1, \ldots, j_k \); see Figure 7.1.

![Figure 7.1: Jobs in period \( l \) in an optimal preemptive schedule.](image)

Note that we must have \( s_{j_i} > 0 \) for \( i = 1, \ldots, k \); otherwise we contradict our assumption that job \( j_i \) is processed in period \( l \) with work volume of \( w_{j_i l} > 0 \). Let \( T_{j_1 j_2} \) be the total processing time of the interval in which jobs \( j_1 \) and \( j_2 \) are processed. Then the speeds of job \( j_1 \) and \( j_2 \) in the interval of length \( T_{j_1 j_2} \) can be determined by the following problem:

\[
\begin{align*}
\text{minimize} & \quad c_l(w_{j_1 l}s_{j_1}^{\alpha-1} + w_{j_2 l}s_{j_2}^{\alpha-1}) \\
\text{subject to} & \quad \frac{w_{j_1 l}}{s_{j_1}} + \frac{w_{j_2 l}}{s_{j_2}} = T_{j_1 j_2}, \quad (7.1b) \\
& \quad s_{j_1}, s_{j_2} > 0. \quad (7.1c)
\end{align*}
\]
For the sake of simplicity, we let \( s_{j_1} = x, s_{j_2} = y, w_{j_1} = a, w_{j_2} = b, \) and \( T_{j_1,j_2} = c. \)

From (7.1b), we obtain that

\[
y = \frac{b}{c - \frac{a}{x}} = \frac{bx}{cx - a}.
\]

Since \( y > 0 \), we must have \( x > a/c \). Now we define

\[
f(x) = ax^{\alpha - 1} + b \left( \frac{bx}{cx - a} \right)^{\alpha - 1}.
\]

Then we have

\[
f'(x) = \frac{a(\alpha - 1)x^{\alpha - 2}[(cx - a)\alpha - b^\alpha]}{(cx - a)^\alpha}.
\]

Let \( f'(x^*) = 0 \). Since \( x > 0 \), we obtain that \( x^* = (a + b)/c \). In addition, when \( a/c < x < x^* \) we have \( f'(x) < 0 \) and when \( x > x^* \) we have \( f'(x) > 0 \). As a result, we know that \( x^* = (a + b)/c \) is a unique global minimum for \( f(x) \) within \((a/c, \infty)\); that is, the optimal solution to (7.1) is \( s_{j_1}^* = s_{j_2}^* = (w_{j_1} + w_{j_2})/T_{j_1,j_2} \). Similarly, for any two adjacent jobs processed in period \( l \), using the same arguments we can prove that their speeds are the same. Since the total work volume in period \( l \) is \( v_l \), we obtain that each job in period \( l \) is processed at the same speed of \( v_l/d_l \) in any optimal preemptive schedule for Problem SMSEC-S-pmtn.

Let \( E_{pmtn} \) be the optimal electricity cost for Problem SMSEC-S-pmtn, and \( W \) the total work volume of all the jobs, i.e., \( W = \sum_{j=1}^{n} w_j \). The following lemma gives the relationship between the work volume, speeds and the selected time periods.

**Lemma 7.2** When periods \( l_1, \ldots, l_m \) are selected to process all the jobs, in any optimal schedule for Problem SMSEC-S-pmtn, the work volume assigned in period \( l_k \) is

\[
v_{l_k}^* = \frac{Wd_{l_k}}{\alpha - \frac{1}{\sqrt{c_{l_k}}} \left( \sum_{i=1}^{m} \frac{d_{i}}{\alpha - \frac{1}{\sqrt{c_{i}}}} \right)},
\]

(7.3)
the speed of each job in period $l_k$ is

$$s^*_l = \frac{W}{\alpha - \sqrt{c_l} \left( \sum_{i=1}^{m} \frac{d_i}{\alpha - \sqrt{c_i} \gamma_i} \right)},$$

(7.4)

and the total electricity cost is

$$W^\alpha \left( \sum_{i=1}^{m} \frac{d_i}{\alpha - \sqrt{c_i}} \right)^{\alpha-1},$$

(7.5)

**Proof**  By Lemma 7.1, we know that there exists an optimal schedule in which each job in period $l_k$ is processed at the same speed of $v_{l_k}/d_{l_k}$. As a result, the electricity cost in period $l_k$ is $c_{l_k} d_{l_k}(v_{l_k}/d_{l_k})^\alpha$, and an optimal schedule can be obtained by solving the following problem:

$$\min_{v_{l_1}, \ldots, v_{l_m}} \sum_{k=1}^{m} c_{l_k} d_{l_k} \left( \frac{v_{l_k}}{d_{l_k}} \right)^\alpha$$

(7.6a)

subject to

$$\sum_{k=1}^{m} v_{l_k} = W,$$

(7.6b)

$$v_{l_k} \geq 0 \text{ for } k = 1, \ldots, m.$$  

(7.6c)

Note that the objective function is convex when $v_{l_m} \geq 0$ and the feasible region is convex, so (7.6) is a convex program. In addition, note that the objective function and the constraints are continuously differentiable.

Let

$$L(v_{l_1}, \ldots, v_{l_m}, \lambda, \mu_1, \ldots, \mu_k) = \sum_{k=1}^{m} c_{l_k} d_{l_k} \left( \frac{v_{l_k}}{d_{l_k}} \right)^\alpha - \lambda \left( \sum_{k=1}^{m} v_{l_k} - W \right) - \sum_{k=1}^{m} \mu_k v_{l_k}.$$
The KKT conditions are:

\[ c_k d_k^{1-\alpha} \alpha(v_k^*)^{\alpha-1} = \lambda^* + \mu_k^* \quad \text{for } k = 1, \ldots, m; \quad (7.7) \]

\[ v_{l_1}^* + v_{l_2}^* + \cdots + v_{l_m}^* = W; \quad (7.8) \]

\[ v_{l_k}^* \geq 0 \quad \text{for } k = 1, \ldots, m; \quad (7.9) \]

\[ \lambda^*, \mu_1^*, \ldots, \mu_m^* \geq 0, \quad (7.10) \]

\[ \mu_k^* v_{l_k}^* = 0 \quad \text{for } k = 1, \ldots, m. \quad (7.11) \]

By the KKT conditions, we must have \( \mu_k^* = 0 \) for \( k = 1, \ldots, m \). Otherwise, suppose \( \mu_h^* > 0 \) for some \( h \in \{1, \ldots, m\} \). Then according to (7.11) we have \( v_{l_h}^* = 0 \). However, from (7.7) we have that \( \lambda^* + \mu_h^* = 0 \), which is impossible when \( \lambda^* \geq 0 \) and \( \mu_h^* > 0 \).

In addition, we must also have \( v_{l_k}^* > 0 \) for \( k = 1, \ldots, m \). Otherwise, suppose \( v_{l_k}^* = 0 \) for some \( h \in \{1, \ldots, m\} \). Then according to (7.7) we have \( \lambda^* = 0 \), and so \( v_{l_k}^* = 0 \) for \( k = 1, \ldots, m \), which contradicts the condition in (7.8). As a result, Equations (7.7)-(7.11) are equivalent to the following equations:

\[ c_k d_k^{1-\alpha} \alpha(v_k^*)^{\alpha-1} = \lambda^* \quad \text{for } k = 1, \ldots, m; \quad (7.12) \]

\[ v_{l_1}^* + v_{l_2}^* + \cdots + v_{l_m}^* = W, \quad (7.13) \]

\[ v_{l_k}^* > 0 \quad \text{for } k = 1, \ldots, m. \quad (7.14) \]

From (7.12) we have

\[ \frac{v_{l_k}^*}{v_{l_j}^*} = \frac{\alpha^{-\sqrt{c_{l_k}}}}{\alpha^{-\sqrt{c_{l_j}}}} \cdot \frac{d_{l_k}}{d_{l_j}}. \]

It is straightforward to check that \( v^* \) as defined in (7.3) is the unique solution that satisfies (7.12)-(7.14). Note that the linear independence constraint qualification (LICQ) for the constrained optimization problem (7.6) holds at \( v^* \), and so the claim follows.

The following results follow easily from Lemma 7.2.
Corollary 7.1 When \( m \) out of \( K \) periods are selected to process the jobs, then in any optimal schedule, these \( m \) periods are selected in nonincreasing order of \( \frac{d_k}{\sqrt[\alpha]{c_k}} \).

Proof By Lemma 7.2, the optimal preemptive schedule using periods \( l_1, \ldots, l_m \) has electricity cost

\[
W^\alpha \left( \sum_{i=1}^{m} \frac{d_{l_i}}{\alpha \sqrt[\alpha]{c_{l_i}}} \right)^{\alpha-1}.
\]

Therefore, we can minimize this cost by choosing the \( m \) periods with the largest values of \( \frac{d_k}{\sqrt[\alpha]{c_k}} \).

\[\blacksquare\]

Theorem 7.1 The optimal electricity cost for Problem SMSEC-S-pmtn is

\[
E_{\text{pmtn}} = \frac{W^\alpha \left( \sum_{i=1}^{K} \frac{d_{l_i}}{\alpha \sqrt[\alpha]{c_{l_i}}} \right)^{\alpha-1}}{\left( \sum_{i=1}^{m} \frac{d_{l_i}}{\alpha \sqrt[\alpha]{c_{l_i}}} \right)^{\alpha-1}}.
\]

Corollary 7.2 In any optimal preemptive schedule for Problem SMSEC-S-pmtn, for any two periods \( l \) and \( k \), we have that

\[
\frac{s^*_l}{s^*_k} = \frac{\alpha-1}{\sqrt[\alpha]{c_k}}.
\]

7.2 NP-hardness of Problem SMSEC-S

Unlike the preemptive speed-scalable scheduling problem we discussed above, the corresponding non-preemptive problem is strongly NP-hard.

Theorem 7.2 Problem SMSEC-S is strongly NP-hard.

Proof Suppose we are given an instance of 3-PARTITION with \( S = \{1, 2, \ldots, 3m\} \) and positive integers \( a_1, \ldots, a_{3m}, b \) such that \( b/4 < a_j < b/2 \) for all \( j \in S \) and \( \sum_{j \in S} a_j = mb \). We construct an instance \( \mathcal{I} \) of Problem SMSEC-S as follows. The TOU tariff is similar to the one depicted in Figure 6.1 in the proof of Theorem 6.1, except the electricity price for each period \([lb+l-1, lb+l]\) for \( l = 1, 2, \ldots, m-1 \) is \( c_l = 2^{a-1} \), and the electricity price for each period \([kb+k, (k+1)b+k]\) for \( k = 0, 1, \ldots, m-1 \) is
$c_k = 1$. For simplicity, we define $\mathcal{L} = \{[lb + l - 1, lb + l) : l = 1, 2, \ldots, m - 1\}$, and $\mathcal{K} = \{[kb + k, (k + 1)b + k) : k = 0, 1, \ldots, m - 1\}$. The number of jobs $n$ is equal to $4m - 1$. The work volume of each job $j$ is as follows:

$$w_j = \frac{1}{2} \quad \text{for } j = 1, 2, \ldots, m - 1;$$

$$w_j = a_{j-m+1} \quad \text{for } j = m, m + 1, \ldots, 4m - 1.$$

We first consider the corresponding optimal preemptive schedule for instance $I$. Note that the total work volume of jobs is $mb + (m - 1)/2$. According to Lemma 7.2, an optimal preemptive schedule can be obtained by solving the following problem:

\begin{align}
\text{minimize} & \quad \sum_{k=0}^{m-1} c_k b \left( \frac{u_k}{b} \right)^\alpha + \sum_{l=1}^{m-1} c_l (v_l)^\alpha \\
\text{subject to} & \quad \sum_{k=0}^{m-1} u_k + \sum_{l=1}^{m-1} v_l = mb + \frac{m - 1}{2}, \tag{7.15b} \\
& \quad u_k \geq 0 \quad \text{for } k = 0, \ldots, m - 1; \tag{7.15c} \\
& \quad v_l \geq 0 \quad \text{for } l = 1, \ldots, m - 1. \tag{7.15d}
\end{align}

where $u_k$ is the work volume assigned to period $k \in \mathcal{K}$, and $v_l$ is the work volume assigned to period $l \in \mathcal{L}$.

According to Lemma 7.2, for each period $k \in \mathcal{K}$, the optimal speed is

$$s_k^* = \frac{mb + \frac{m-1}{2}}{\alpha \sqrt{\frac{b}{\sqrt{\alpha}} + \sum_{l=1}^{m-1} \frac{1}{\sqrt{\alpha} \sqrt{2\alpha - 1}}}} = 1,$$

and the optimal work volume is $u_k^* = b$. In addition, for each period $l \in \mathcal{L}$, the optimal speed is $s_l^* = 1/2$, and the optimal work volume is $v_l^* = 1/2$. The total electricity cost is $E_{\text{pmtn}} = mb + (m - 1)/2$.

Let $E^*$ be the optimal electricity cost for instance $I$ of Problem SMSEC-S, i.e. the electricity cost of an optimal non-preemptive schedule for instance $I$. Then we
have $E^* \geq E_{\text{pmtn}}$. By Lemma 7.2, we know that the above solution $(u^*, v^*)$ is the only feasible solution that achieves an objective function value of $E_{\text{pmtn}}$.

Now, suppose we have a non-preemptive schedule $\sigma_1$ for instance $I$ with electricity cost $E^* = E_{\text{pmtn}}$. Let $u_k$ and $v_l$ be the work volume assigned to period $k \in K$ and $l \in L$ in schedule $\sigma_1$, respectively. Then the work volume assigned to each period in this non-preemptive schedule is the same as in the optimal preemptive schedule. Otherwise, suppose $(u, v) \neq (u^*, v^*)$. We construct a preemptive schedule $\sigma_2$ such that the work volume assigned to each period is given as in $(u, v)$. Let $E^2_{\text{pmtn}}$ be the optimal electricity cost generated by $\sigma_2$. Then we have $E^* \geq E^2_{\text{pmtn}} > E_{\text{pmtn}}$, which contradicts our assumption that $E^* = E_{\text{pmtn}}$.

In addition, in the non-preemptive schedule $\sigma_1$, the jobs processed within the same period must be processed at the same speed. Otherwise, suppose in some period $h \in P$, the jobs are not processed at the same speed. Then we can construct a preemptive schedule $\sigma_3$ for the jobs in period $h$ as follows. By Lemma 7.1 we know that the electricity cost in period $h$ is minimized when all jobs are processed at the same speed in a preemptive schedule, say $s^*_h$. Suppose this is the case in $\sigma_3$. Let $E^h$ and $E^h_{\text{pmtn}}$ be the optimal electricity cost of schedule $\sigma_1$ and $\sigma_3$ in period $h$, respectively. Then we have $s^*_h = 1$ and $E^h_{\text{pmtn}} = c_kb = b$ when $h \in K$, and $s^*_h = 1/2$ and $E^h_{\text{pmtn}} = c_l(1/2)^\alpha = 1/2$ when $h \in L$. By Lemma 7.1, we also know that the above solution $s^*_h$ is the only feasible solution that achieves an objective function of $E^h_{\text{pmtn}}$. As a result, we have $E^h > E^h_{\text{pmtn}}$ for any period $h$ in which the jobs are not processed at the same speed, which contradicts our assumption that $E^* = E_{\text{pmtn}}$. Note that in an optimal preemptive schedule, $s^*_k = 1$ for $k \in K$, which is not equal to $s^*_l = 1/2$ for $l \in L$. By the non-preemptive restriction on jobs for Problem SMSEC-S and since $a_1, \ldots, a_{3m}$ are integers, we obtain that each job in $\sigma_1$ can only be processed within one period. This can be done if and only if every job $j \in \{1, 2, \ldots, m - 1\}$ can be processed between time periods $[lb + l - 1, lb + l)$ for $l = 1, 2, \ldots, m - 1$, and the remaining jobs can be partitioned over the $m$ intervals of length $b$, which can be done
if and only if 3-PARTITION has a solution. Since 3-PARTITION is strongly NP-hard (Garey and Johnson 1979), the theorem follows.

7.3 Inapproximability of Problem SMSEC-S

In this section, we first investigate the structural properties of optimal non-preemptive schedules under two-period and three-period TOU tariff schemes. Then we show that there is no PTAS for Problem SMSEC-S, using a reduction from PARTITION.

For a given two-period TOU tariff scheme with durations $d_1$ and $d_2$, we let

$$\xi = 2^\alpha \left( \frac{d_2}{d_1} \right)^{\alpha-1} \left[ \left( 1 + \frac{\sum_{i \in J} w_i}{\min_{i \in J} \{w_i\}} \right)^\alpha - \left( \frac{\sum_{i \in J} w_i}{\min_{i \in J} \{w_i\}} \right)^\alpha \right],$$

and $\gamma = \max \{ 2^\alpha - 1, \xi \}$.

Lemma 7.3 Suppose the given TOU tariff scheme has only two periods with durations $d_1, d_2$ and electricity prices $c_1, c_2$. If $c_2 > \gamma c_1$, then in any optimal schedule no job is processed in both periods.

Proof We will prove this lemma by contradiction. Suppose in an optimal schedule $\sigma_1$ there exists a job $j$ that is processed in both periods as in Figure 7.2.

![Figure 7.2: Job j is processed in both periods in an optimal schedule.](image)

For simplicity, we denote the processing time of job $j$ in periods 1 and 2 as $t_1$ and $t_2$ respectively, and denote its speed in $\sigma_1$ as $s_j$. Without loss of generality, we assume that all the other jobs processed in period 1 are jobs 1 to $k$, and their total
processing time is $t$. Using techniques similar those in the proof of Lemma 7.2, we know that jobs 1 to $k$ are processed at the same speed, say $s$. In addition, we let $w_{j1}$ and $w_{j2}$ be the work volume of job $j$ assigned to period 1 and 2 respectively, and $w$ be the total work volume of jobs 1 to $k$. It is easy to see that $w_{j1} = w_j t_1/(t_1 + t_2)$ and $w_{j2} = w_j t_2/(t_1 + t_2)$.

Case 1: $t_1 \geq t_2$. In this case, we construct a new schedule $\sigma_2$, in which we keep the position of job $j$ in period 1 unchanged, and process the entire of job $j$ with a uniform speed $w_j/t_1$ in this position, while keeping the positions of all other jobs unchanged. We denote the electricity cost for processing job $j$ in $\sigma_i$ as $E_j^i$ for $i = 1, 2$. Then we have

$$E_j^1 = c_1 t_1 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha + c_2 t_2 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha$$

and

$$E_j^2 = c_1 t_1 \left( \frac{w_j}{t_1} \right)^\alpha.$$

In what follows, we will prove that $E_j^1 > E_j^2$, i.e.,

$$c_2 t_2 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha > c_1 t_1 \left( \frac{w_j}{t_1} \right)^\alpha - c_1 t_1 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha.$$

Since $c_2 > (2^\alpha - 1)c_1$, we will prove the following inequality instead.

$$2^\alpha - 1 \geq \frac{t_1}{t_2} \left[ \left( \frac{1 + t_2}{t_1} \right)^\alpha - 1 \right].$$

We let $t_2/t_1 = x$, and define $f(x) = [(1 + x)^\alpha - 1]/x$. Since $t_1 \geq t_2$, we have that $0 < x \leq 1$. The derivative of $f(x)$ is $f'(x) = [\alpha x (1 + x)^{\alpha - 1} - (1 + x)^\alpha + 1]/x^2$. Let $g(x) = \alpha x (1 + x)^{\alpha - 1} - (1 + x)^\alpha + 1$. Then we have $g'(x) = \alpha x (\alpha - 1) (1 + x)^{\alpha - 2} \geq 0$, so $g(x) \geq g(0) = 0$ for $0 < x \leq 1$. That is, $f'(x) \geq 0$ for $0 < x \leq 1$, and so we have $f(x) \leq f(1) = 2^\alpha - 1$. As a result, we obtain that $E_j^1 > E_j^2$, which contradicts schedule $\sigma_1$ being optimal.

Case 2: $t_1 < t_2$. In this case, we construct a new schedule $\sigma_3$, in which we process jobs 1 to $k$ and the entirety of job $j$ in period 1 at the same speed $s'$, while keeping
the positions of all the other jobs in period 2 unchanged. Denote the total electricity of jobs 1 to \( k \) and job \( j \) in schedule \( \sigma_i \) as \( E^i \) for \( i = 1, 3 \). Then the total electricity cost of jobs 1 to \( k \) and job \( j \) is

\[
E^1 = c_1 s^\alpha + c_1 t_1 s_j^\alpha + c_2 t_2 s_j^\alpha = c_1 t \left( \frac{w}{t} \right)^\alpha + c_1 t_1 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha + c_2 t_2 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha.
\]

We also have

\[
E^3 = c_1 d_1 (s')^\alpha = c_1 d_1 \left( \frac{w + w_j}{d_1} \right)^\alpha.
\]

In what follows, we will prove that \( E^1 > E^3 \), i.e.,

\[
c_2 t_2 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha > c_1 d_1 \left( \frac{w + w_j}{d_1} \right)^\alpha - c_1 t \left( \frac{w}{t} \right)^\alpha - c_1 t_1 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha.
\]

Note that \( t_1 < t_2 \), so we have

\[
\frac{t_2}{t_1 + t_2} > \frac{1}{2}.
\]

Since \( t_2 \leq d_2 \), we obtain that

\[
c_2 t_2 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha = c_2 w_j \frac{t_2}{t_1 + t_2} \left( \frac{w_j}{t_1 + t_2} \right)^{\alpha - 1} > \frac{1}{2} c_2 w_j \left( \frac{w_j}{2t_2} \right)^{\alpha - 1} \geq \frac{1}{2} c_2 w_j \left( \frac{w_j}{2d_2} \right)^{\alpha - 1}.
\]

Since \( t \leq d_1 \), we have

\[
c_1 d_1 \left( \frac{w + w_j}{d_1} \right)^\alpha - c_1 t \left( \frac{w}{t} \right)^\alpha - c_1 t_1 \left( \frac{w_j}{t_1 + t_2} \right)^\alpha \leq c_1 d_1 \left( \frac{w + w_j}{d_1} \right)^\alpha - c_1 t \left( \frac{w}{t} \right)^\alpha
\]

\[
\leq c_1 d_1 \left( \frac{w + w_j}{d_1} \right)^\alpha - c_1 d_1 \left( \frac{w}{d_1} \right)^\alpha.
\]

As a result, in order to prove \( E^1 > E^3 \), we only need to prove that

\[
c_2 > 2^\alpha \left( \frac{d_2}{d_1} \right)^{\alpha - 1} \left[ \left( \frac{w + w_j}{w_j} \right)^\alpha - \left( \frac{w}{w_j} \right)^\alpha \right] c_1.
\]
Let $h(x) = (1 + x)\alpha - x^\alpha$. Then we have $h'(x) = \alpha[(1 + x)^{\alpha-1} - x^{\alpha-1}] > 0$ for $x \geq 0$. Note that $w/w_j \leq (\sum_{i \in \mathcal{J}} w_i)/(\min_{i \in \mathcal{J}} w_i)$. As a result, we have

$$h(0) \leq h\left(\frac{w}{w_j}\right) \leq h\left(\frac{\sum_{i \in \mathcal{J}} w_i}{\min_{i \in \mathcal{J}} w_i}\right).$$

Since $c_2 > \xi c_1$, we have that $E^1 > E^3$, which contradicts schedule $\sigma_1$ being optimal. ■

**Lemma 7.4** Suppose the given TOU tariff scheme has only two periods with durations $d_1, d_2$ and electricity prices $c_1, c_2$. If $c_2 > \gamma c_1$, then in any optimal non-preemptive schedule all of the jobs must be processed in only period 1.

**Proof** By Lemma 7.3, we know that there are three possible ways to process the jobs in an optimal non-preemptive schedule: process all the jobs in period 1, process all the jobs in period 2, or process a subset of jobs in period 1 with total work volume $W_1$, and the others in period 2 with total work volume $W_2$. We denote the total electricity cost for the above three possible schedules as $E_1, E_2, E_3$, respectively. Then we have

$$E_1 = c_1 d_1 \left(\frac{W_1 + W_2}{d_1}\right)^\alpha,$$

$$E_2 = c_2 d_2 \left(\frac{W_1 + W_2}{d_2}\right)^\alpha,$$

$$E_3 = c_1 d_1 \left(\frac{W_1}{d_1}\right)^\alpha + c_2 d_2 \left(\frac{W_2}{d_2}\right)^\alpha.$$

In what follows, we will prove that $E_1 < E_2$ and $E_1 < E_3$.

Note that $E_2/E_1 = (c_2/c_1)(d_1/d_2)^{\alpha-1} > 2^\alpha$, so we have $E_1 < E_2$. To see that $E_1 < E_3$, we need to prove that

$$c_2 > \left(\frac{d_2}{d_1}\right)^{\alpha-1}\left[\left(\frac{W_1 + W_2}{W_2}\right)^\alpha - \left(\frac{W_1}{W_2}\right)^\alpha\right]c_1.$$

Using a similar argument to the one in the proof of Lemma 7.3, since $c_2 > \gamma c_1$, the above inequality holds. ■
Lemma 7.5 Suppose the given TOU tariff scheme has only three periods with durations $d_1, d_2, d_3$ and electricity prices $c_1, c_2, c_3$, in which $d_1 = d_3$ and $c_1 = c_3$. If $c_2 > \gamma c_1$, then in any optimal non-preemptive schedule no job is processed in period 2. In addition, if there are more than 2 jobs, then the work volume of jobs assigned to periods 1 and 3 must greater than 0.

Proof By Lemma 7.3 and Lemma 7.4, we know that there is no job processed in period 2 in an optimal non-preemptive schedule. As a result, there are only three possible ways to process the jobs in an optimal non-preemptive schedule: process all the jobs in period 1, process all the jobs in period 3, or process a subset of jobs in period 1 with total work volume $W_1$, and the others in period 3 with total work volume $W_3$. We denote the total electricity cost for the above three possible schedules as $E_1, E_2,$ and $E_3$, respectively. Then we have

$$E_1 = E_3 = c_1 d_1 \left( \frac{W_1 + W_3}{d_1} \right)^{\alpha},$$

and

$$E_2 = c_1 d_1 \left( \frac{W_1}{d_1} \right)^{\alpha} + c_3 d_3 \left( \frac{W_3}{d_3} \right)^{\alpha}.$$

Since $d_1 = d_3$ and $c_1 = c_3$, in order to prove that $E_2 < E_1 = E_3$, we need to prove that $W_1^{\alpha} + W_3^{\alpha} < (W_1 + W_3)^{\alpha}$, which is always true when $\alpha > 1$. \hfill \blacksquare

Theorem 7.3 There is no PTAS for Problem SMSEC-S, unless $P = NP$.

Proof The proof is based on a reduction from PARTITION to Problem SMSEC-S. The PARTITION problem is described as follows: Given a set $S = \{1, 2, \ldots, n\}$ and positive integers $a_1, a_2, \ldots, a_n$, does there exist a subset $I$ of $S$ such that $\sum_{j \in I} a_j = \sum_{j \not\in I} a_j$.

Given any instance $I_1$ of PARTITION, we construct an instance $I_2$ of Problem SMSEC-S with a three-period TOU tariff scheme as follows. The durations are $d_1 = d_2 = d_3 = (\sum_{j \in S} a_j)/2$. For simplicity, we let $S = (\sum_{j \in S} a_j)/2$. The electricity prices are $c_1 = c_3 = 1/(2S)$ and $c_2 = \gamma c_1$. The number of jobs is $n$. The work volume
for each job \( j \) is \( w_j = a_j \), and the power consumption is \( q_j = s^\alpha \). Without loss of
generality, we assume that \( d_1 = d_2 = d_3 = S \) is an integer; otherwise there is no
solution for \( I_1 \).

Note that when there exists a solution to \( I_1 \), the optimal electricity cost is

\[
E_{opt}^1 = c_1d_1\left(\frac{S}{d_1}\right)^\alpha + c_3d_3\left(\frac{S}{d_3}\right)^\alpha = \frac{1}{2}\left(\frac{S}{S}\right)^\alpha + \frac{1}{2}\left(\frac{S}{S}\right)^\alpha = 1.
\]

When there does not exist a solution to \( I_1 \), without loss of generality, we assume
that in an optimal non-preemptive schedule the total work volume of jobs assigned
to period 1 is \( S + x \), and the corresponding total work volume of jobs in period 3 is
\( S - x \) for some integer \( 1 \leq x \leq S - 1 \). So

\[
E_{opt}^2 = c_1d_1\left(\frac{S+x}{d_1}\right)^\alpha + c_3d_3\left(\frac{S-x}{d_3}\right)^\alpha \geq \frac{1}{2}\left[\left(1 + \frac{x}{S}\right)^\alpha + \left(1 - \frac{x}{S}\right)^\alpha\right] \geq \frac{1}{2}\left[\left(1 + \frac{1}{S}\right)^\alpha + \left(1 - \frac{1}{S}\right)^\alpha\right].
\]

Suppose there is a PTAS for Problem SMSEC-S; that is, for any error parameter
\( \varepsilon > 0 \), we can find a schedule in time polynomial in the input size whose cost is within
a factor of \( (1 + \varepsilon) \) of the minimum cost. (Note that the running time does not need
to be polynomial in \( 1/\varepsilon \).) For simplicity, we let \( \delta = (1/2)[(1 + 1/S)^\alpha + (1 - 1/S)^\alpha] \).

Note that \( \delta \) is dependent on the instance \( I_1 \), and \( \delta > 1 \) for all \( S \geq 1 \). On instance \( I_2 \),
we set the error parameter to \( \varepsilon = (1/2)(\delta - 1) \), and run the PTAS. Now, the solution
produced will have electricity cost \( E_{algo} \leq (1 + \varepsilon)E_{opt}^1 = 1 + \varepsilon \) when there exists a
solution to \( I_1 \), and have electricity cost \( E_{algo} \geq E_{opt}^2 > 1 + \varepsilon \) when there does not
exist a solution to \( I_1 \). As a result, we can conclude whether there is a solution to
any instance of PARTITION in polynomial time by looking at the output of a PTAS
using an error parameter of \( (1/2)(\delta - 1) \). Therefore, there exists no PTAS for any
instance of Problem SMSEC-S, unless \( P = NP \).
7.4 Structural properties of optimal schedules for Problem SMSEC-S with $|\mathcal{P}| = 2$

Before proposing some approximation algorithms for Problem SMSEC-S, we examine how the optimal schedules look like in some special cases of Problem SMSEC-S. We show that the optimal schedules for Problem SMSEC-S highly depend on the relationship between electricity prices and the value of $\alpha$ even in the simple case when the durations of time periods and the work volume of jobs are the same. To illustrate, we consider two special cases in which $|\mathcal{P}| = 2$ and $d_1 = d_2 = d$ in Sections 7.4.1 and 7.4.2. Without loss of generality, we assume that $c_1 < c_2$.

7.4.1 Case 1: single job

In this subsection, we consider the structural properties of optimal schedules when there is only a single job with work volume $w$.

**Lemma 7.6** If the job is processed in only one period, then in an optimal schedule, it will be processed in period 1 at speed $s = w/d$.

**Proof** Suppose the job is processed in some period $k \in \{1, 2\}$. If its processing time is less than $d$, then we can slow down the speed of the job until it reaches the end of the period. Since $c_1 < c_2$, in an optimal schedule the job is processed in period 1.

**Lemma 7.7** In any optimal schedule under the above TOU tariffs, the job must be processed at speed $w/d$ or $w/(2d)$.

**Proof** By Lemma 7.6, we know that $w/s \geq d$, i.e., $s \leq w/d$. In addition, since there are two periods, we have $w/s \leq 2d$, i.e., $s \geq w/(2d)$. Then in order to minimize the total electricity cost for processing the job, we need to solve the following problem:

\[
\text{minimize } f(s) = c_1 ds^\alpha + c_2 \left(\frac{w}{s} - d\right)s^\alpha \\
\text{subject to } \frac{w}{2d} \leq s \leq \frac{w}{d}.
\]
Then we have $f'(s) = (c_1 - c_2)ds^{\alpha-1} + (\alpha - 1)c_2ws^{\alpha-2} = s^{\alpha-2}[(c_1 - c_2)d\alpha s + (\alpha - 1)c_2w]$.

Let $f'(s^*) = 0$. Since $s > 0$, we obtain that $s^* = ((\alpha - 1)c_2w)/((c_2 - c_1)\alpha d)$. When $s < s^*$, we have $f'(s) > 0$ and when $s > s^*$, we have $f'(s) < 0$; that is, $s^*$ is a global maximum for the function in (7.16) over $(0, +\infty)$.

Since $w/(2d) \leq s \leq w/d$, as a result, the minimum value of $f(s)$ can only be obtained when $s = w/d$ or $s = w/(2d)$.

Let $s^*$ be the global maximum for the function in (7.16) over $(0, +\infty)$ as in the proof of Lemma 7.7. Now we can obtain the optimal speed for processing the job according to the relationship between $s^*$, $w/d$ and $w/(2d)$.

**Theorem 7.4** When $s^* > w/d$, i.e., $c_1 < c_2 < c_1\alpha$, then in an optimal schedule, the job is processed at a speed of $w/(2d)$.

**Proof** Since $s^* > w/d$, we know that $f(s)$ is monotonically increasing within $[w/(2d), w/d]$. That is, we obtain a minimum electricity cost at speed $w/(2d)$.

**Theorem 7.5** When $1 < \alpha < 2$ and $s^* < w/(2d)$, i.e., $c_2 > c_1\alpha/(2 - \alpha)$, then in an optimal schedule, the job is processed at a speed of $w/d$.

**Proof** Since $s^* < w/(2d)$, we know that $f(s)$ is monotonically decreasing within $[w/(2d), w/d]$, and so we obtain an optimal schedule at $s = w/d$. Note that when $\alpha \geq 2$, it is impossible to have $(\alpha - 2)c_2 < -c_1\alpha$, and so $s^* \geq w/(2d)$ always holds.

**Theorem 7.6** When $\alpha \geq 2$ and $w/(2d) \leq s^* \leq w/d$, i.e., $c_2 \geq c_1\alpha$, then in an optimal schedule, the job is processed at a speed of

a. $w/(2d)$ when $c_1\alpha \leq c_2 < (2^\alpha - 1)c_1$;

b. either $w/(2d)$ or $w/d$ when $c_2 = (2^\alpha - 1)c_1$;

c. $w/d$ when $c_2 > (2^\alpha - 1)c_1$.  

Proof Let $E_1$ be the total electricity cost when the job is processed only in period 1, i.e. its speed is $w/d$ and $E_1 = c_1 d(w/d)^{\alpha}$. Let $E_2$ be the total electricity cost when the job is processed in both periods, i.e., its speed is $w/(2d)$ and $E_2 = c_1 d(w/(2d))^{\alpha} + c_2 d(w/(2d))^{\alpha}$.

Note that when $c_2 = (2^\alpha - 1)c_1$ we have $E_1 = E_2$; when $c_2 < (2^\alpha - 1)c_1$ we have $E_1 < E_2$; and when $c_2 > (2^\alpha - 1)c_1$ we have $E_1 > E_2$. Since $\alpha > 1$, we always have $\alpha < 2^\alpha - 1$, that is, $c_1 \alpha < (2^\alpha - 1)c_1$.

Theorem 7.7 When $1 < \alpha < 2$ and $w/(2d) \leq s^* \leq w/d$, i.e., $c_1 \alpha \leq c_2 \leq c_1 \alpha (2 - \alpha)$, then in an optimal schedule, the job is processed at a speed of

- $w/(2d)$ when $c_1 \alpha \leq c_2 < (2^\alpha - 1)c_1$;
- either $w/(2d)$ or $w/d$ when $c_2 = (2^\alpha - 1)c_1$;
- $w/d$ when $(2^\alpha - 1)c_1 < c_2 < \alpha/(2 - \alpha)c_1$.

Proof The proof is similar to the one for Theorem 7.6. We only need to prove that $2^\alpha - 1 < \alpha/(2 - \alpha)$ when $1 < \alpha < 2$.

Let $g(x) = (2^x - 1) - x/(2 - x)$. Since $g(1) = 0$, in order to prove that $g(x) < 0$ for $1 < x < 2$, we only need to prove that $g'(x) < 0$ when $1 < x < 2$.

Note that $g'(x) = (2^x \ln 2 - 2)/(2 - x)^2$. Let $r(x) = 2^x \ln 2(2 - x)^2$, then we have $r'(x) = 2^x \ln 2(2 - x)[\ln 2(2 - x) - 2] < 0$ when $1 < x < 2$. As a result, we have $r(x) < r(1) = \ln 2 < 2$ for any $x \in (1, 2)$, or in other words, $g'(x) < 0$. As a result, we have $2^\alpha - 1 < \alpha/(2 - \alpha)$ when $1 < \alpha < 2$.

7.4.2 Case 2: two jobs with equal work volume

In this subsection, we consider the structural properties of optimal schedules when we have two jobs with equal work volume $w$. Without loss of generality, we assume that in any optimal schedule, these two jobs are scheduled in the order of $(1, 2)$. 
Lemma 7.8 If job 1 and job 2 are processed in only one period, then in an optimal schedule, they are processed in period 1 at speed $2w/d$.

Proof According to Lemma 7.1, the speeds of job 1 and 2 are the same when they are processed in only one period. Since they have the same work volume $w$, the speeds of job 1 and 2 are $2w/d$.

Lemma 7.9 If these two jobs are processed in two periods, then in an optimal schedule, the total processing time of the two jobs must be equal to $2d$.

Proof Similar to the proof of Lemma 7.6.

Lemma 7.10 If the two jobs are processed in two periods, in which job 1 is processed at speed $s_1$ and job 2 is processed at speed $s_2$, then in an optimal schedule we have $s_1 \geq s_2$.

Proof By contradiction. Suppose we have $s_1 < s_2$. Then the corresponding schedule $\sigma_1$ is:

As a result, we have $w/s_1 \geq d$. Let $E_1$ be the total electricity cost for schedule $\sigma_1$. Then we have

$$E_1 = c_1 ds_1^\alpha + c_2 \left( \frac{w}{s_1} - d \right) s_1^\alpha + c_2 \frac{w}{s_2} s_2^\alpha = c_1 ds_1^\alpha + c_2 \left( \frac{w}{s_1} - d \right) s_1^\alpha + c_2 \left( 2d - \frac{w}{s_1} \right) s_2^\alpha,$$

where $w/s_1 + w/s_2 = 2d$.

Now let us process job 1 at speed $s_2$, and process job 2 at speed $s_1$, and call this new schedule $\sigma_2$. The schedule $\sigma_2$ looks like:

Let $E_2$ be the corresponding electricity cost for schedule $\sigma_2$. Then we have

$$E_2 = c_1 \frac{w}{s_2} s_2^\alpha + c_1 \left( \frac{w}{s_1} - d \right) s_1^\alpha + c_2 ds_1^\alpha = c_1 \left( 2d - \frac{w}{s_1} \right) s_2^\alpha + c_1 \left( \frac{w}{s_1} - d \right) s_1^\alpha + c_2 ds_1^\alpha.$$
Therefore,

\[ E_1 - E_2 = (c_2 - c_1) \left( 2d - \frac{w}{s_1} \right) (s_2^\alpha - s_1^\alpha). \]

Since \( s_2 > s_1 \), we obtain that \( E_1 > E_2 \), which is a contradiction.

**Lemma 7.11** When the two jobs are processed in two periods, then in an optimal schedule job 2 will be processed at a speed of \( s^* \), where \( w/(2d) < s^* < w/d \).

**Proof** By Lemma 7.10, we know that \( w/s_1 \leq d \). Then in order to minimize the total electricity cost, we need to solve the following problem:

\[
\begin{align*}
\text{minimize} & \quad f(s_1, s_2) = c_1 \frac{w}{s_1} s_1^\alpha + c_1 \left( d - \frac{w}{s_1} \right) s_2^\alpha + c_2 ds_2^\alpha \\
\text{subject to} & \quad \frac{w}{s_1} + \frac{w}{s_2} = 2d; \\
& \quad \frac{w}{2d} < s_2 \leq \frac{w}{d}.
\end{align*}
\]  

From (7.19), we know that \( s_1 = ws_2/(2ds_2 - w) \) and so \( f(s_1, s_2) = f(s_2) = c_1 w((ws_2)/(2ds_2 - w))^{\alpha - 1} + c_1((w/s_2) - d)s_2^\alpha + c_2 ds_2^\alpha \).

Note that

\[ f'(s_2) = \left[ c_1 w(\alpha - 1) + (c_2 - c_1) d \alpha s_2 - c_1 w(\alpha - 1) \left( \frac{w}{2ds_2 - w} \right)^\alpha \right] s_2^{\alpha - 2}. \]

so we have \( f'(s_2)|_{s_2 = w/(2d) + \varepsilon} < 0 \) for any sufficiently small \( \varepsilon > 0 \) and \( f'(s_2)|_{s_2 = w/d} > 0 \) when \( c_1 < c_2 \). That is, in an optimal schedule job 2 will be processed at a speed of \( s^* \), where \( w/(2d) < s^* < w/d \).

From Sections 7.4.1 and 7.4.2, we know that the speeds of jobs and the order of jobs in an optimal schedule for Problem SMSEC-S rely on the relationship between electricity prices and the value of \( \alpha \).
7.5 Approximation algorithms for Problem SMSEC-S

In this section, we propose and analyze approximation algorithms for Problem SMSEC-S by transforming a preemptive schedule for Problem SMSEC-S-pmtn into a non-preemptive one for Problem SMSEC-S. We saw in Section 7.1 that in any optimal preemptive schedule, the work volume and the speeds of jobs in each period $l$ do not depend on the jobs, only the duration and electricity price of period $l$. Therefore, without loss of generality, we assume that the jobs are processed in the order $(1, 2, \ldots, n)$ in an optimal preemptive schedule. We let $\sigma_{pmtn}$ be an optimal preemptive schedule, and denote the corresponding non-preemptive schedule generated by Algorithm 7.5.1 (below) as $\sigma_{nonpmtn}$.

Algorithm 7.5.1 A $\sum_{k=1}^{K} (c_k / \min_k \{c_k\})^{1/\alpha}$-approximation algorithm for Problem SMSEC-S

Require: $w_j, q_j = s_j^* \text{ for } j \in J, c_k, d_k \text{ for } k \in P$.
1: Create an optimal preemptive schedule $\sigma_{pmtn}$ with the jobs ordered $(1, 2, \ldots, n)$ as in Section 7.1; that is, all $K$ periods are selected to process all of the jobs, and the work volume and the speeds of jobs $s_1^*, \ldots, s_K^*$ in each period are set as in Lemma 7.2.
2: Let $T_j$ be the total processing time of job $j$ in $\sigma_{pmtn}$.
3: for $j = 1 \text{ to } n$ do
4: \hspace{1em} Keep the position of job $j$ unchanged, and process job $j$ with uniform speed, i.e. $s_j' = w_j/T_j$. \\
5: end for

Theorem 7.8 Algorithm 7.5.1 is a $\sum_{k=1}^{K} (c_k / \min_k \{c_k\})^{1/\alpha}$-approximation algorithm for Problem SMSEC-S.

Proof Denote the electricity cost for processing job $j$ in $\sigma_{nonpmtn}$ and $\sigma_{pmtn}$ as $E_{nonpmtn}^j$ and $E_{pmtn}^j$ respectively. In what follows we will prove that $E_{nonpmtn}^j \leq \sum_{k=1}^{K} (c_k / \min_k \{c_k\})^{1/\alpha} E_{pmtn}^j$ for any job $j \in J$. 

Let $b_j$ be the period in which job $j$ begins processing, $e_j$ the period in which job $j$ ends processing, and $s_k^*$ the speed of job $j$ in period $k$ in $\sigma_{pmtn}$. Let $T_{jb_j}$ and $T_{je_j}$ be the processing time of job $j$ in period $b_j$ and $e_j$ respectively. Then we obtain that

$$E_{pmtn}^j = c_{b_j}T_{jb_j}(s_{b_j}^*)^\alpha + \sum_{k=b_j+1}^{e_j-1} c_k d_k(s_k^*)^\alpha + c_{e_j}T_{je_j}(s_{e_j}^*)^\alpha,$$

$$E_{nonpmtn}^j = c_{b_j}T_{jb_j}(s_{b_j}^')^\alpha + \sum_{k=b_j+1}^{e_j-1} c_k d_k(s_k^*)^\alpha + c_{e_j}T_{je_j}(s_{e_j}^')^\alpha.$$

Note that $(a + b)/(c + d) \leq a/c + b/d$ for any $a, b, c, d > 0$. Therefore,

$$\frac{E_{nonpmtn}^j}{E_{pmtn}^j} = \frac{c_{b_j}T_{jb_j}(s_{b_j}^')^\alpha + \sum_{k=b_j+1}^{e_j-1} c_k d_k(s_k^')^\alpha + c_{e_j}T_{je_j}(s_{e_j}^')^\alpha}{c_{b_j}T_{jb_j}(s_{b_j}^*)^\alpha + \sum_{k=b_j+1}^{e_j-1} c_k d_k(s_k^*)^\alpha + c_{e_j}T_{je_j}(s_{e_j}^*)^\alpha} \leq \frac{c_{b_j}T_{jb_j}(s_{b_j}^')^\alpha}{c_{b_j}T_{jb_j}(s_{b_j}^*)^\alpha} + \sum_{k=b_j+1}^{e_j-1} \frac{c_k d_k(s_k^')^\alpha}{c_k d_k(s_k^*)^\alpha} + \frac{c_{e_j}T_{je_j}(s_{e_j}^')^\alpha}{c_{e_j}T_{je_j}(s_{e_j}^*)^\alpha}$$

$$= \left(\frac{s_{b_j}^'}{s_{b_j}^*}\right)^\alpha + \sum_{k=b_j+1}^{e_j-1} \left(\frac{s_{k}^'}{s_k^*}\right)^\alpha + \left(\frac{s_{e_j}^'}{s_{e_j}^*}\right)^\alpha.$$

Let $\max_k \{s_k^*\}$ be the maximum speed in an optimal preemptive schedule. Since

$$T_{jb_j}s_{b_j}^* + \sum_{k=b_j+1}^{e_j-1} d_k s_k^* + T_{je_j}s_{e_j}^* = T_js_j',$$

we have

$$s_j' \leq \max \{s_k^* : b_j \leq k \leq e_j\} \leq \max_k \{s_k^*\}.$$
Note that a job can at most be processed in $K$ periods, so
\[
\frac{E_j^{\text{nonpmtn}}}{E_j^{\text{pmtn}}} \leq \sum_{k=1}^{K} \left( \frac{s_j^l}{s_k^l} \right)^{\alpha} \leq \sum_{k=1}^{K} \left( \frac{\max_k \{ s_k^* \}}{s_k^*} \right)^{\alpha}.
\]

By Corollary 7.2 we know that $s_j^* / s_k^* = \sqrt[\alpha]{c_k / c_l}$. As a result, we have
\[
\frac{E_{\text{nonpmtn}}}{E_{\text{opt}}} \leq \frac{E_{\text{nonpmtn}}}{E_{\text{pmtn}}} \leq \sum_{k=1}^{K} \left( \frac{c_k}{\min_k \{ c_k \}} \right)^{\alpha-1},
\]
where $E_{\text{opt}}$ is the minimum electricity cost for Problem SMSEC-S.

By Theorem 7.1 we know that all of the $K$ periods will be selected to process the jobs in an optimal preemptive schedule. However, when constructing a non-preemptive schedule, as shown in Lemma 7.4, it may be better to only select a subset of all the $K$ periods to process the jobs. The following lemma describes the relationship between the optimal cost when using a subset of periods and the optimal cost when using all the periods for Problem SMSEC-S-pmtn.

**Lemma 7.12** Suppose $m$ out of $K$ periods are optimally selected to process all of the jobs in a preemptive schedule. Let $l_1, \ldots, l_m$ be the $m$ selected periods, and $l'_1, \ldots, l'_{K-m}$ be the $K - m$ periods that are not selected. Let $E_m$ be the optimal cost obtained by using only $m$ periods for $m = 1, \ldots, K$. Then we have $E_m \leq (K/m)^{\alpha-1} E_K$.

**Proof** By Lemma 7.2 we have the following result.
\[
\frac{E_m}{E_K} = \frac{E_m}{E_{\text{pmtn}}} = \left( \frac{\sum_{i=1}^{m} \frac{d_{l_i}}{\sqrt[\alpha]{c_{l_i}}}}{\sum_{k=1}^{K} \frac{d_k}{\sqrt[\alpha]{c_k}}} \right)^{1-\alpha}.
\]

Since $1 - \alpha < 0$, in order to prove the lemma, we only need to prove that
\[
\frac{\sum_{i=1}^{m} \frac{d_{l_i}}{\sqrt[\alpha]{c_{l_i}}}}{\sum_{k=1}^{K} \frac{d_k}{\sqrt[\alpha]{c_k}}} \geq \frac{m}{K}.
\]
This is equivalent to proving that

\[(K - m) \left( \sum_{i=1}^{m} \frac{d_{ij}}{\alpha \cdot \sqrt{c_i}} \right) \geq m \left( \sum_{j=1}^{K-m} \frac{d_{ij}'}{\alpha \cdot \sqrt{c_{j}''}} \right).\]

By Corollary 7.1 we know that the m periods are selected according to nonincreasing value of \(d_k / \alpha \cdot \sqrt{c_k}\) in a preemptive schedule; that is,

\[\frac{d_{ij}'}{\alpha \cdot \sqrt{c_{j}''}} \leq \min_{i=1, \ldots, m} \left\{ \frac{d_{ik}}{\alpha \cdot \sqrt{c_{i}}} \right\} \text{ for any } j = 1, \ldots, K - m.\]

The lemma follows.

For simplicity, when m periods are selected to process all of the jobs in a preemptive schedule, we denote the set of the selected m periods as \(P_m\), the corresponding optimal preemptive schedule as \(\sigma_{pmtn}^m\), and the associated electricity cost \(E_{pmtn}^m\). Note that these m selected periods in \(P_m\) may not be consecutive, and as a result, we cannot process the jobs as we did in Algorithm 7.5.1. Based on Lemma 7.12, we propose Algorithm 7.5.2 below, in which we process the entirety of each job \(j\) in the period in which job \(j\) has maximum processing time in \(\sigma_{pmtn}^m\). We denote \(\sigma_{nonpmtn}^m\) as the non-preemptive schedule generated by Algorithm 7.5.2. Let \(E_{nonpmtn}^m\) be the corresponding electricity cost of schedule \(\sigma_{nonpmtn}^m\), and let \(E_{pmtn}^m,j\) and \(E_{nonpmtn}^m,j\) be the electricity cost of job \(j\) in their preemptive and non-preemptive schedules respectively.

**Theorem 7.9** Algorithm 7.5.2 is a \(\max_{1 \leq m \leq K} \beta_m\)-approximation algorithm for Problem SMSEC-S, where

\[\beta_m = mK^{\alpha - 1} \left( \frac{\max_{k \in P_m} \{c_k\}}{\min_{k \in P_m} \{c_k\}} \right)^{\alpha - 1} \text{ for } m = 1, \ldots, K.\]

**Proof** We will prove that \(E_{nonpmtn}^m,j \leq \beta_m E_{pmtn}^m,j\) for any \(j \in J\) and \(m \in \{1, \ldots, K\}\), which indicates that the approximation ratio of Algorithm 7.5.2 is at most \(\max_{1 \leq m \leq K} \beta_m\). Suppose job \(j\) is processed in the following \(n_j\) (\(n_j \leq m\)) periods: \(\{l_1, \ldots, l_{n_j}\} \subset P_m\); see Figure 7.4.
Algorithm 7.5.2 A $\max_{1 \leq m \leq K} \beta_m$-approximation algorithm for Problem SMSEC-S

Require: $w_j, q_j = s_j^\alpha$ for $j \in \mathcal{J}$, $c_k, d_k$ for $k \in \mathcal{P}$.

1: for $m = 1$ to $K$ do
2: Determine the set $\mathcal{P}_m$ according to nonincreasing order of $d_k / \sqrt[\alpha]{c_k}$.
3: Create an optimal preemptive schedule $\sigma^m_{\text{pmtn}}$ according to the set $\mathcal{P}_m$ with the jobs ordered $(1, 2, \ldots, n)$.
4: for $j = 1$ to $n$ do
5: Let $T_{jk}$ be the processing time of job $j$ in period $k$ in $\sigma^m_{\text{pmtn}}$.
6: Calculate $k' = \arg \max_{k \in \mathcal{P}_m} \{T_{jk}\}$.
7: Keep the position of job $j$ in period $k'$ unchanged, and process the entire job $j$ in period $k'$ with uniform speed $s_j'$.
8: end for
9: Denote the above non-preemptive schedule as $\sigma^m_{\text{nonpmtn}}$, calculate its corresponding electricity cost $E^m_{\text{nonpmtn}}$.
10: end for
11: Calculate $m^* = \arg \min_{1 \leq m \leq K} \{E^m_{\text{nonpmtn}}\}$, choose schedule $\sigma^m_{\text{nonpmtn}}$ to process the jobs.

Let $T_{jl}$ be the maximum processing time of job $j$ within the $n_j$ periods, and $s^*_l$ the optimal speed of the selected period $l_k$ in an optimal preemptive schedule $\sigma^m_{\text{pmtn}}$. According to Algorithm 7.5.2, we will keep the position of job $j$ in period $l_k'$ unchanged, and process the entire job $j$ in this position. Then we obtain that $E^m_{\text{pmtn}} = \sum_{i=1}^{n_j} c_i T_{jl_i} (s^*_i)^\alpha$ and $E^m_{\text{nonpmtn}} = c_{l'} T_{jl'} (s_{l'}')^\alpha$. Since

$$\sum_{i=1}^{n_j} T_{jl_i} s^*_i = T_{jl'} s_{l'}',$$
and \( T_{j_{l'}} \geq T_{j_l} \) for \( i = 1, \ldots, n_j \), we have
\[
\begin{align*}
\sum_{i=1}^{n_j} s_{i}^{*} & \leq n_j \max_{k \in \mathcal{P}_m} \{ s_{i}^{*} \} \\
\sum_{i=1}^{n_j} s_{i}^{*} & \leq n_j \max_{k \in \mathcal{P}_m} \{ s_{i}^{*} \} \\
\end{align*}
\]

Note that \( c_{l'} T_{j_{l'}} \leq \sum_{i=1}^{n_j} c_{l_i} T_{j_{l_i}} \), and so we obtain that
\[
\begin{align*}
\frac{E_{m, j}^{\text{nonpmtn}}}{E_{m, j}^{\text{pmtn}}} & \leq \frac{c_{l'} T_{j_{l'}} (m \max_{k \in \mathcal{P}_m} \{ s_{k}^{*} \})^{\alpha}}{\sum_{i=1}^{n_j} c_{l_i} T_{j_{l_i}} (\min_{k \in \mathcal{P}_m} \{ s_{k}^{*} \})^{\alpha}} \\
& \leq m^{\alpha} \left( \frac{\max_{k \in \mathcal{P}_m} \{ s_{k}^{*} \}}{\min_{k \in \mathcal{P}_m} \{ s_{k}^{*} \}} \right)^{\alpha}. \\
\end{align*}
\]

By Lemma 7.2 we know that
\[
\frac{\max_{k \in \mathcal{P}_m} \{ s_{k}^{*} \}}{\min_{k \in \mathcal{P}_m} \{ s_{k}^{*} \}} = \left( \frac{\max_{k \in \mathcal{P}_m} \{ c_{k} \}}{\min_{k \in \mathcal{P}_m} \{ c_{k} \}} \right)^{\frac{1}{\alpha-1}}.
\]

Therefore,
\[
\frac{\max_{k \in \mathcal{P}_m} \{ s_{k}^{*} \}}{\min_{k \in \mathcal{P}_m} \{ s_{k}^{*} \}} = \left( \frac{\max_{k \in \mathcal{P}_m} \{ c_{k} \}}{\min_{k \in \mathcal{P}_m} \{ c_{k} \}} \right)^{\frac{\alpha}{\alpha-1}},
\]

and so we have
\[
\frac{E_{m, j}^{\text{nonpmtn}}}{E_{\text{opt}}} \leq \frac{E_{m, j}^{\text{nonpmtn}}}{E_{K}} \leq \frac{E_{m, j}^{\text{nonpmtn}}}{E_{m}} \cdot \left( \frac{K}{m} \right)^{\alpha-1} \leq m^{\alpha} \left( \frac{\max_{k \in \mathcal{P}_m} \{ c_{k} \}}{\min_{k \in \mathcal{P}_m} \{ c_{k} \}} \right)^{\frac{\alpha}{\alpha-1}} \left( \frac{K}{m} \right)^{\alpha-1} = \beta_{m},
\]

where \( E_{\text{opt}} \) is the minimum electricity cost for Problem SMSEC-S, and \( E_{m} \) is the minimum electricity cost obtained by using only \( m \) periods for Problem SMSEC-S-pmtn.

The main idea of Algorithm 7.5.1 is to transform the optimal preemptive schedule \( \sigma_{\text{pmtn}} \) into a non-preemptive schedule \( \sigma_{\text{nonpmtn}} \) while maintaining the processing time of each job. However, some jobs in \( \sigma_{\text{nonpmtn}} \) have to be processed in the periods with (extremely) high electricity prices. In order to exploit the variable pricing of electricity, we only choose a subset of periods to process all of the jobs in Algorithm 7.5.2. There
are still shortcomings in Algorithm 7.5.2 since the processing time of each job is shortened in the corresponding non-preemptive schedule. Based on the observations from Lemma 7.4, let us examine how Algorithms 7.5.1 and 7.5.2 perform in some specific cases. Let $|\mathcal{J}| = 1$, and consider a two-period TOU tariff scheme in which $d_1 = d_2 = d$. For simplicity, let $E_{\text{algo}1}$ and $E_{\text{algo}2}$ be the total electricity cost generated by Algorithms 7.5.1 and 7.5.2 for the given instance respectively.

**Example 7.1** Let $c_2 = c_1$. From Algorithms 7.5.1 and 7.5.2, we obtain that $E_{\text{algo}1} = c_1(2d)(W/(2d))^\alpha$, and $E_{\text{algo}2} = c_1d(W/d)^\alpha$. As a result, we have $E_{\text{algo}2}/E_{\text{algo}1} = 2^{\alpha-1}$. That is, Algorithm 7.5.1 performs better than Algorithm 7.5.2 in this case.

**Example 7.2** Let $c_2 = \gamma c_1$, where $\gamma = 2^\alpha(2^\alpha - 1)$. From Algorithms 7.5.1 and 7.5.2, we obtain that $E_{\text{algo}1} = c_1d(W/(2d))^\alpha + c_2d(W/(2d))^\alpha$, and $E_{\text{algo}2} = c_1d(W/d)^\alpha$. As a result, we have

$$\frac{E_{\text{algo}1}}{E_{\text{algo}2}} = \frac{1 + \gamma}{2^\alpha} = \frac{4^\alpha - 2^\alpha + 1}{2^\alpha} > 1.$$ 

That is, Algorithm 7.5.2 performs better than Algorithm 7.5.1 in this case.

For some special cases, we may use the advantages of both algorithms to obtain better results. When the electricity tariffs satisfy $d_1/\sqrt[\alpha]{c_1} \geq \cdots \geq d_K/\sqrt[\alpha]{c_K}$ (e.g. when $d_k = d$ for $k \in \mathcal{P}$, and $c_k$ is nondecreasing), we know that $\mathcal{P}_m = \{1, 2, \ldots, m\}$ for any $1 \leq m \leq K$. As a result, when $m$ periods are selected to process all of the jobs, these periods are consecutive, and so there always exists a preemptive schedule in which jobs are processed consecutively. Therefore, we propose the following algorithm.

The following theorem follows directly from Theorem 7.8 and Lemma 7.12.

**Theorem 7.10** Algorithm 7.5.3 is a $\min_{1 \leq m \leq K} \left\{ \left(\frac{K}{m}\right)^{\alpha-1} \sum_{k=1}^{m} \left(\frac{c_k}{\min_{1 \leq k \leq m}(c_k)}\right)^{\frac{1}{\alpha}} \right\}$-approximation algorithm for Problem SMSEC-S with $d_1/\sqrt[\alpha]{c_1} \geq \cdots \geq d_K/\sqrt[\alpha]{c_K}$. 

### 7.6 Experimental study

Since Algorithms 7.5.1, 7.5.2 and 7.5.3 are all algorithms for Problem SMSEC-S that have non-constant performance guarantees that depend on the number of periods
Algorithm 7.5.3 An approximation algorithm for Problem SMSEC-S with $d_1/\sqrt[\alpha]{c_1} \geq \cdots \geq d_K/\sqrt[\alpha]{c_K}$

| Require: | $w_j, q_j = s_j^q$ for $j \in J$, $c_k, d_k$ for $k \in P$. |
| 1: | Calculate $m^* = \arg \min_{1 \leq m \leq K} \left\{ \left( \frac{K}{m} \right)^{\alpha-1} \sum_{k=1}^{m} \left( \frac{c_k}{\min_{k \leq m} \{c_k\}} \right)^{\alpha-1} \right\}$. |
| 2: | Create an optimal preemptive schedule $\sigma_{pmtn}^m$ according to the set of $P_{m^*} = \{1, 2, \ldots, m^*\}$. |
| 3: | for $j = 1$ to $n$ do |
| 4: | Keep the position of job $j$ unchanged, and process job $j$ with a uniform speed, i.e., $s_j' = w_j/T_j$. |
| 5: | end for |

as well as the electricity prices, it is useful to empirically test their performance on randomly generated instances. In this section, we conduct two experiments. The first experiment determines the impact of the ratio of electricity prices on the behavior of Algorithms 7.5.1, 7.5.2 and 7.5.3. The second experiment compares the performance of Algorithms 7.5.1 and 7.5.2. For simplicity, we let $\alpha = 3$ in our experiments. Since it is difficult to obtain the optimal electricity cost for these instances, we use the corresponding optimal preemptive electricity cost $E_{pmtn}^m$ as a lower bound on the optimal non-preemptive electricity cost $E_{opt}$ when evaluating the performance of these algorithms. In addition, for different instances, it does not make much sense to compare the electricity cost of the schedules they generate directly, so we use the ratio of electricity costs to compare performance.

We use the Python programming language to implement Algorithms 7.5.1, 7.5.2 and 7.5.3, and calculate the electricity costs for the instances on a computer with 1.6GHz Intel Core i5 processor and 2GB of RAM running the OS X operating system.

7.6.1 Experiment 1: ratio of electricity cost vs. ratio of electricity prices

In this experiment, we will illustrate the impact of the ratio of electricity prices under a two-period TOU tariff scheme in which $c_1 \leq c_2$. We construct 50 randomly generated instances as follows: in each instance, an integer duration $d_k$ is generated from the uniform distribution on $\{1, \ldots, 20\}$ for $k \in \{1, 2\}$. The number of jobs $n$ and
an integer work volume $w_j$ for each job $j$ are generated from the uniform distribution on \{1, \ldots, 20\}. All of these quantities are generated independently, except that we require $d_2 \leq d_1$ to let $d_1/\sqrt[\alpha]{c_1} \geq d_2/\sqrt[\alpha]{c_2}$, so that we can implement Algorithm 7.5.3. For simplicity, we define $\theta = c_2/c_1$, which we constrain to be either 1, 4, 64, 256, 2048, or 32768. For each randomly generated instance, we calculate the ratios between the total electricity costs obtained by Algorithm 7.5.1, 7.5.2, 7.5.3 and the corresponding optimal preemptive schedule for all the different values of the ratio of electricity prices $\theta$. For each instance, the CPU time for calculating the corresponding schedules is less than 30 seconds.

Table 7.1 (on page 100) shows the average ratio of total electricity costs obtained by Algorithms 7.5.1, 7.5.2, 7.5.3 and the corresponding optimal preemptive schedule for $\theta \in \{1, 4, 64, 256, 2048, 32768\}$, respectively. As we mentioned before, we let $E_{\text{algo1}}$, $E_{\text{algo2}}$ and $E_{\text{algo3}}$ denote the total electricity cost generated by Algorithms 7.5.1, 7.5.2 and 7.5.3, respectively. From Table 7.1, we can see that when $\theta$ is small (i.e. $\theta = 1, 4$), Algorithm 7.5.1 performs better than Algorithm 7.5.2. When $\theta$ becomes large enough, Algorithm 7.5.1 can perform really poorly, while Algorithm 7.5.2 obtains schedules with near-optimal electricity costs. This observation matches our theoretical results in Lemmas 7.3 and 7.4. In addition, Algorithm 7.5.3 performs as well as Algorithm 7.5.1 when $\theta = 1$. However, when $\theta = 4$, Algorithm 7.5.3 performs worse than both Algorithms 7.5.1 and 7.5.2. When $\theta$ becomes larger, Algorithm 7.5.3 performs similarly to Algorithm 7.5.2.

To illustrate, we also compare the electricity costs obtained by Algorithms 7.5.1, 7.5.2, 7.5.3 and the corresponding optimal preemptive schedule directly for a specific combination of $n = 6, d_k, w_j$ while varying $\theta \in \{1, 2, 4, 8, 16, 32, 64\}$; see Figure 7.5. From this figure, we see that when $\theta$ is small, Algorithm 7.5.1 performs better than Algorithm 7.5.2. When $\theta$ becomes larger enough, the electricity cost obtained by Algorithm 7.5.1 increases dramatically, while Algorithms 7.5.2 and 7.5.3 have a near-optimal electricity cost.
7.6.2 Experiment 2: performance evaluation of Algorithms 7.5.1 and 7.5.2

From Experiment 1, we observe that the ratio of electricity prices plays an important role in the performance of Algorithms 7.5.1, 7.5.2 and 7.5.3. Since Algorithm 7.5.3 can only be implemented under special structures in which $d_1/\sqrt[n]{c_1} \geq \cdots \geq d_K/\sqrt[n]{c_K}$, in this subsection, we will only evaluate the performance of Algorithms 7.5.1 and 7.5.2 under general instances, using the following combinations for the number of periods $K$ and the number of jobs $n$:

$$\{(K, n) : K \in \{2, 6, 10, 20, 30, 50\}, n \in \{5, 10, 20, 40, 70, 150\}\}.$$

We randomly generate 50 instances for each combination $(K, n)$. In each instance, for each period an integer duration $d_k$ is generated from the uniform distribution on $\{1, \ldots, 20\}$, and an electricity price $c_k$ is generated from the uniform distribution on $[0.05, 1]$ for $k = 1, \ldots, K$. In addition, an integer work volume $w_j$ is generated from the uniform distribution on $\{1, \ldots, 20\}$ for $j = 1, \ldots, n$. All quantities are generated independently. Note that $0.05 \leq c_k \leq 1$ for $k = 1, \ldots, K$; that is, the ratio
of electricity prices between any two different periods can be at most 20. This is reasonable, since most of the TOU tariffs are pre-determined legislatively by the states within a fixed range of prices to ensure the stability of the retail electricity market and reduce customer resistance on price uncertainty (Braithwait et al. 2007). For each instance, the CPU time for calculating the corresponding schedule is less than 30 seconds.

Tables 7.2 and 7.3 show the ratio between the electricity costs obtained by Algorithms 7.5.1 and 7.5.2 and the corresponding optimal preemptive schedule, respectively. We observe that the maximum average ratio between the electricity costs obtained by Algorithm 7.5.1 and the optimal preemptive schedule is 1.375 while the same ratio for Algorithm 7.5.2 is 32.287. Table 7.4 shows the ratio between the electricity costs obtained by Algorithms 7.5.1 and 7.5.2. From Table 7.4 we see that the average ratios between electricity cost obtained by Algorithm 7.5.2 and Algorithm 7.5.1 are all greater than 1 for all the combinations \((K, n)\), which suggests that Algorithm 7.5.1 always outperforms Algorithm 7.5.2 in our empirical testing, and that although the performance guarantee of Algorithm 7.5.1 is non-constant, it provides good results in practice when the ratios of electricity prices between different periods are within a small bounded range.

In addition, we find that the number of periods and the number of jobs also affect the performance of these two algorithms: when the number of jobs is fixed, the larger the number of periods, the higher the ratio of electricity costs between the schedules generated by these algorithms and the corresponding optimal preemptive schedule; see Figure 7.6; when the number of periods is fixed, the larger the number of jobs, the lower the ratio of electricity costs between the schedules generated by these algorithms and the corresponding optimal preemptive schedule; see Figure 7.7.
Figure 7.6.: Average ratio between electricity costs obtained by Algorithms 7.5.1, 7.5.2 and their corresponding optimal preemptive schedule when we fix $n = 5$.

Figure 7.7.: Average ratio between electricity costs obtained by Algorithms 7.5.1, 7.5.2 and their corresponding optimal preemptive schedule when we fix $K = 50$.

### 7.6.3 Tables

Table 7.1: Average ratio between electricity costs obtained by Algorithm 7.5.1, 7.5.2, 7.5.3 and the corresponding optimal preemptive schedule.

<table>
<thead>
<tr>
<th>Average ratio of electricity costs</th>
<th>Statistical results</th>
<th>$\theta = c_2/c_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$E_{algo1}/E_{pmtn}$</td>
<td>Mean</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>1.000</td>
</tr>
<tr>
<td>$E_{algo2}/E_{pmtn}$</td>
<td>Mean</td>
<td>1.167</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.271</td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>2.575</td>
</tr>
<tr>
<td>$E_{algo3}/E_{pmtn}$</td>
<td>Mean</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Table 7.2: Average ratio between electricity costs obtained by Algorithm 7.5.1 and the optimal preemptive schedule.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Statistical results</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Mean</td>
<td>1.053</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.092</td>
</tr>
<tr>
<td>6</td>
<td>Mean</td>
<td>1.145</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.138</td>
</tr>
<tr>
<td>10</td>
<td>Mean</td>
<td>1.211</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.130</td>
</tr>
<tr>
<td>20</td>
<td>Mean</td>
<td>1.308</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.142</td>
</tr>
<tr>
<td>30</td>
<td>Mean</td>
<td>1.363</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.160</td>
</tr>
<tr>
<td>50</td>
<td>Mean</td>
<td>1.375</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.094</td>
</tr>
</tbody>
</table>

Table 7.3: Average ratio between electricity costs obtained by Algorithm 7.5.2 and the optimal preemptive schedule.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Statistical results</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Mean</td>
<td>1.298</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.369</td>
</tr>
<tr>
<td>6</td>
<td>Mean</td>
<td>2.338</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.750</td>
</tr>
<tr>
<td>10</td>
<td>Mean</td>
<td>3.926</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>1.894</td>
</tr>
<tr>
<td>20</td>
<td>Mean</td>
<td>10.384</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>3.779</td>
</tr>
<tr>
<td>30</td>
<td>Mean</td>
<td>15.339</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>4.864</td>
</tr>
<tr>
<td>50</td>
<td>Mean</td>
<td>32.287</td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>14.796215</td>
</tr>
</tbody>
</table>
Table 7.4: Average ratio between electricity costs obtained by Algorithm 7.5.2 and Algorithm 7.5.1.

<table>
<thead>
<tr>
<th>K</th>
<th>Statistical results</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>70</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1.226</td>
<td>1.091</td>
<td>1.118</td>
<td>1.033</td>
<td>1.020</td>
<td>1.014</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.284</td>
<td>0.116</td>
<td>0.127</td>
<td>0.039</td>
<td>0.026</td>
<td>0.019</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2.032</td>
<td>1.491</td>
<td>1.272</td>
<td>1.149</td>
<td>1.085</td>
<td>1.047</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.561</td>
<td>0.252</td>
<td>0.133</td>
<td>0.078</td>
<td>0.052</td>
<td>0.027</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>3.234</td>
<td>1.886</td>
<td>1.461</td>
<td>1.238</td>
<td>1.154</td>
<td>1.075</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.423</td>
<td>0.350</td>
<td>0.151</td>
<td>0.081</td>
<td>0.044</td>
<td>0.025</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>8.060</td>
<td>3.397</td>
<td>2.016</td>
<td>1.481</td>
<td>1.265</td>
<td>1.134</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.032</td>
<td>0.859</td>
<td>0.258</td>
<td>0.120</td>
<td>0.066</td>
<td>0.032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>11.528</td>
<td>5.306</td>
<td>2.676</td>
<td>1.724</td>
<td>1.418</td>
<td>1.265</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.116</td>
<td>1.342</td>
<td>0.382</td>
<td>0.120</td>
<td>0.071</td>
<td>0.035</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>23.871</td>
<td>10.360</td>
<td>4.410</td>
<td>2.393</td>
<td>1.713</td>
<td>1.337</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.788</td>
<td>3.562</td>
<td>0.849</td>
<td>0.209</td>
<td>0.111</td>
<td>0.048</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part IV

Conclusion
8. CONCLUDING REMARKS AND FUTURE DIRECTIONS

To the best of our knowledge, this dissertation is one of the first to consider (1) a multi-objective flow shop scheduling problem with traditional time-based objectives (i.e. makespan) as well as energy-based objectives (i.e. peak power consumption) and (2) a single machine scheduling problem with the objective of minimizing total electricity cost under time-of-use tariffs. In particular, we studied the permutation flow shop problem with peak power consumption constraints (the PFSPP problem) and the single machine scheduling problem with electricity costs (the SMSEC problem).

We proposed two integer programming formulations and accompanying valid inequalities for the PFSPP problem with discrete speeds (Problem PFSPP-DU). A key feature of our formulations is variables and constraints that keep track of jobs running concurrently across machines. This may be of interest in other applications (e.g. Thörnblad 2013). We investigated the computational performance of these formulations with instances arising from the manufacturing of cast iron plates. Although our valid inequalities for the assignment and positional formulation resulted in better computational performance, especially for small-to-moderate sized instances, we still had difficulty obtaining optimal schedules in a reasonable amount of time for instances with large numbers of jobs and machines. One potential direction for future research is to develop stronger valid inequalities for our models, in the hopes of strengthening these models and improving their computational performance.

We also showed that the PFSPP problem can be recast as an asymmetric TSP when the flow shop has two machines with zero intermediate storage (Problems PFSPP-DTZ and PFSPP-CTZ). In addition, we were able to obtain stronger structural characterizations of optimal schedules and polynomial time algorithms to find these
schedules when the speed set is continuous and the work volume of jobs satisfy certain conditions (Problem PFSPP-CTZ).

For the SMSEC problem, we considered the problem when jobs can only be processed on a uniform-speed machine (i.e. Problems SMSEC-U, SMSEC-U-pmtn and SMSEC-U-pyr) and when jobs can be processed on a speed-scalable machine (i.e. Problems SMSEC-S-pmtn and SMSEC-S). We showed that Problem SMSEC-U is strongly NP-hard and in fact inapproximable within a constant factor unless $P = NP$. We also gave an exact polynomial-time algorithm for Problem SMSEC-U-pyr in which all the jobs have the same work volume and the electricity prices follow a so-called pyramidal structure, and showed that Problem SMSEC-S is strongly NP-hard and in fact has no polynomial time approximation scheme. We then proposed and analyzed different approximation algorithms for Problem SMSEC-S (i.e. Algorithms 7.5.1, 7.5.2 and 7.5.3) and empirically tested their performance on randomly generated instances.

Of course, there are many possible directions for future research stemming from this dissertation. For the PFSPP problem, when the number of machines is greater than three, solving this bicriteria problem can have a heavy computational cost, since it is already NP-hard to find an optimal schedule when only minimizing the makespan without additional energy- or power-related criteria. However, when the number of machines is equal to two, the complexity of the PFSPP problem is still open; the single objective problem can be solved in polynomial time. It would be interesting to fully characterize which two-machine variants of the PFSPP problem are NP-hard or polynomial time solvable. It would also be interesting to consider different time or energy objectives (e.g. total weighted completion time, carbon footprint) or some other complex machine environments with peak power consumption constraints. For the SMSEC problem, an interesting, natural open question is whether there exists a constant-factor approximation algorithm for Problem SMSEC-S. It would also be interesting to consider minimizing total electricity costs under TOU tariffs for other scheduling environments.
LIST OF REFERENCES
LIST OF REFERENCES


K. Fang, N.A. Uhan, F. Zhao, and J.W. Sutherland. Scheduling on a single machine under time-of-use tariffs, 2013b. working paper.


A. Sharma, F. Zhao, and J.W. Sutherland. Ecological scheduling of a manufacturing enterprise operating under a time-of-use electricity tariff, 2013. working paper.


VITA
Kan Fang was born in Taizhou, China in 1985. He received B.S. in mathematics and applied mathematics from Zhejiang University in 2007. After that, he received M.S. in operations research from Zhejiang University in 2009. Since 2009, he is pursuing the Ph.D. degree in School of Industrial Engineering at Purdue University and is expected to receive the degree by December 2013. His research mainly focuses on scheduling, combinatorial optimization, and integer programming.

The following contains a list of journals and conference papers that has been published during Mr. Fang’s Ph.D. research at Purdue University.

