The Pennsylvania State University

The Graduate School

The Harold and Inge Marcus Department of Industrial and Manufacturing Engineering

FLOW SHOP SCHEDULING WITH SYNCHRONOUS AND ASYNCHRONOUS TRANSPORTATION TIMES

A Dissertation in

Industrial Engineering and Operations Research

by

Kwei-Long Huang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2008

The dissertation of Kwei-Long Huang was reviewed and approved* by the following:

*Signatures are on file in the Graduate School

ABSTRACT

This research studies two flow shop scheduling problems which consider transportation times between machines. The first problem considers a special case of a two-machine flow shop scheduling problem with asynchronous transportation times and the second one consider an application of flow shop scheduling to automated manufacturing cells with a synchronous material transportation device. The objective of both problems is to find a job schedule which minimizes the makespan – the completion time of the last job.

In the first problem, not only transportation times are explicitly provided but also the availability of the transporter is considered. In the model, there is one transporter with a specific capacity to transport jobs from the first machine to the second machine. The processing times on the first machine are job-independent. A threshold value for the transporter's capacity is derived. When the capacity of the transporter is greater than or equal to the threshold value, a dynamic programming algorithm is developed to obtain an optimal schedule. Given that n is the number of jobs, the computational effort of the proposed dynamic programming algorithm is shown to be $O(n^3)$, which is better than the best algorithm found in the literature.

This research also considers a new flow shop scheduling problem with synchronous material movement. The automated machining center consists of a loading/unloading (L/U) station, m processing machines, and a rotary table. The L/U station and the processing machines surround the rotary table. In this machining center, a job is first loaded onto the rotary table at the L/U station. Then, the table rotates to transfer the job to the first machine, and subsequently to the second machine and so on. After being processed by the m machines, the job is transported back to the L/U station where it is unloaded from the machining center. A rotation of the table occurs only when all stations are finished with their jobs, including the loading and unloading operations at the L/U station. The mechanism of transferring all these jobs on the rotary table simultaneously to their next stations is referred to as synchronous material movement.

Regarding the machining center with synchronous material movement, the simplest model with a single machine is studied first. The problem is shown to be NP-hard in the strong sense. A polynomial time algorithm is developed for a special case which assumes a constant unloading or loading time for all jobs. Moreover, due to the systematic structure of the problem, a dynamic programming algorithm is provided to obtain an optimal solution for a generalized version of the problem with *m* machines. The computational effort of the dynamic programming algorithm is also presented.

Two-phase heuristic algorithms are developed to solve the problems with one machine and two machines. For each problem, two constructive heuristics and the modified neighborhood search algorithm are proposed, and computational experiments are conducted to test the performance of the proposed algorithms. The experimental results show that the two-phase algorithms generate high quality solutions in a very short time. A tighter lower bound is also developed for each case.

TABLE OF CONTENTS

LIST OF FIGURES vii

LIST OF TABLES ix

ACKNOWLEDGEMENTS x

Chapter 1 Introduction and Overview 1

1.1 Introduction 1
1.2 Research Problems 3
1.3 Research Objectives and Contributions 5
1.4 Thesis Overview 7

Chapter 2 Literature Review 10

2.1 Scheduling 10
2.2 Flow Shop 13
2.3 Job Shop 21
2.4 Flow Shop with Transfer Times and Transporters 24
2.5 Automated Manufacturing Cell 30
2.6 Computational Complexity Theory 34

Chapter 3 Two-machine Flow Shop with Transportation Considerations 38

3.1 Introduction 38
3.2 Dynamic Programming Algorithm for $TF_2 \mid p_{j1} \equiv p_1, v = 1, c \geq u \mid C_{\max}$ 40
3.3 Complexity Analysis 47
3.4 Concluding Summary 50

Chapter 4 Flow Shop with Synchronous Material Movement 51

4.1 Introduction 51
4.2 T-line Machining Center with One CNC Machine 54
4.3 Special Case – Constant Unloading Time 65
4.4 Dynamic Programming Algorithm for $F_1 \mid synmv, re - LU \mid C_{\max}$ 72
4.5 Two-Phase Heuristic Algorithm for One CNC Machine 78
4.6 Computational Results 93
4.7 Concluding Summary 98

Chapter 5 Two-machine Flow Shop with Synchronous Material Movement 100

5.1 Introduction 100
5.2 Dynamic Programming Algorithm for $F_2 \mid synmv, re - LU \mid C_{\max}$ 102
5.3 Heuristic Algorithm for $F_2 \mid synmv, re - LU \mid C_{\max}$ 107

5.4 Computational Results....116
5.5 Dynamic Programming Algorithm for $F_m \mid synmv, re-LU \mid C_{\max}$ 121
5.6 Concluding Summary....132

Chapter 6 Summary and Future Study....133

6.1 Summary....133
6.2 Future Research....136

BIBLIOGRAPHY....140

APPENDIX Gilmore-Gomory Algorithm....147

LIST OF FIGURES

Figure 1.1. A T-line machining center with *m* machines 5

Figure 2.1. Workflow in a general flow shop (Source: Baker, 1995)....................14

Figure 2.2. Workflow in a job shop (Source: Baker, 1995) 22

Figure 2.3. Robot-centered cell layout (Logendran and Srisjandarajah 1996)....................32

Figure 2.4. Mobile robot cell layout (Logendran and Srisjandarajah 1996) 32

Figure 2.5. In-line robot cell layout (Logendran and Srisjandarajah 1996) 32

Figure 2.6. Complexity hierarchies of deterministic scheduling problems:....................37

Figure 3.1. Batch *i* containing *u*+1 jobs transported to machine 2 at d_i....................43

Figure 3.2. Batch *i* is split into two small batches 43

Figure 3.3. An example with *k*=20 and *j*=10 49

Figure 4.1. Examples of Cincinnati Milacron T-line machining centers....................52

Figure 4.2. A T-line machining center with two CNC machines 53

Figure 4.3. A T-line machining center with one CNC machine 55

Figure 4.4. Schedule of jobs at each station in a one-CNC T-line machining center....................56

Figure 4.5. Schedule σ on the L/U station and CNC machine 59

Figure 4.6. A Jb-type job is sequenced in the first position 61

Figure 4.7. A Jc-type job is sequenced in the first position 61

Figure 4.8. A Ja-type job is sequenced in the second position....................62

Figure 4.9. A Jb-type job is sequenced in the second position....................62

Figure 4.10. A Ja-type job is sequenced in the third position 62

Figure 4.11. A Jc-type job is sequenced in the third position 63

Figure 4.12. Idle time on the L/U station in the third cycle when $a_{[1]} + b_{[1]} < c_{[1]}$ 64

Figure 4.13. Idle time on the CNC machine in the third cycle when $a_{[1]} + b_{[1]} > c_{[1]}$ 64

Figure 4.14. Schedule in a one-machine T-line machining center with constant unloading time.65

Figure 4.15. Two-machine flow shop problem with blocking for $J_{[j]}, 3 \le j \le n$ 66

Figure 4.16. The makespan *Z'* for the modified problem 67

Figure 4.17. J_0 is not sequenced in the first position for Problem *G* 68

Figure 4.18. J_0 is sequenced in the first position for Problem *G* 68

Figure 4.19. J_{n+1} is not sequenced in the last position for Problem *G*..69

Figure 4.20. J_{n+1} is sequenced in the last position for Problem *G* 69

Figure 4.21. The makespan for the problem $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$ 70

Figure 5.1. Schedule of jobs at each station in a two-CNC T-line machining center................101

Figure 5.2. Schedule of jobs at each station in an *m*-CNC T-line machining center.................126

Figure 5.3. Schedule of jobs for the relaxed problem with 4 machines (*h*=6) 129

Figure 5.4. Schedule of jobs for the relaxed problem with 4 machines (*h*=7) 129

Figure 5.5. Schedule of jobs for the relaxed problem with 4 machines (*h*=8) 130

LIST OF TABLES

Table 4.1: Number of operations required for each stage 73

Table 4.2: Job data for a one-machine T-line machining center 75

Table 4.3: Job data for the example of CAS....................82

Table 4.4: Job data for the example of CAI 86

Table 4.5: Experimental design and data generating rules for $F_1 \mid synmv, re-LU \mid C_{\max}$ 94

Table 4.6. Summary of the average makespans obtained by the DP, heuristics, and LB 95

Table 4.7: Summary of average relative errors from optimum and LB....................96

Table 5.1. Number of operations required for each stage 103

Table 5.2. Job data for a two-CNC T-line machining center 105

Table 5.3: Job data for the example of CAGI....................112

Table 5.4: Experimental design and data generating rules for $F_2 \mid synmv, re-LU \mid C_{\max}$ 117

Table 5.5: Summary of the average makespans obtained by the DP, heuristics, and LB....................118

Table 5.6: Summary of average relative errors from optimum and LB....................119

Table 5.7. Number of operations required for each stage 123

Table 5.8: Job data for the example of the algorithm to obtain LB_R 128

Table 5.9: Job data for the example after sorting in descending order 128

Table A.1: Job data for the Gilmore-Gomory Algorithm 148

Table A.2: Sorted processing times and the cost for each edge 149

Table A.3: The optimal tour 150

ACKNOWLEDGEMENTS

I would like to express my enormous appreciation to my adviser and committee chair, Dr. José A. Ventura, for his invaluable guidance, support, and encouragement throughout this research. Without his constructive advice and criticism, this dissertation would not be well shaped. His attitudes towards research and commitment to quality have greatly influenced me. I would also like to thank my committee members, Dr. A. Ravi Ravindran, Dr. Terry P. Harrison, and Dr. Tao Yao for their insightful suggestions and their time for this work.

I would like to thank my classmate, Zheng Jia, for working with me on the DP project which is the basis of Chapter 3. I am also very grateful to my friend, Wenny Chandra, for proofreading the final draft of my dissertation. As an international student, I would like to offer my heartfelt gratitude to these people who have helped me during my studies at Penn State, especially to my friends in BAOCP.

Most of all, I express my deepest thanks to my parents Mu-Syong Huang (黃木雄) and Siou-Lian Peng (彭秀連). Without their unreserved support and consistent encouragement, I would not have been able to pursue my dream and accomplish the achievement. To them this thesis is dedicated.

Chapter 1

Introduction and Overview

1.1 Introduction

Due to the rapid advance in technology and the prevalence of e-commerce, global competition is not only a trend but a pressure situation for a company. In order to maintain a competitive advantage, companies attempt to optimize not only plant operations but also the interplant activities between facilities. For plant operations, accurate scheduling plays an important role in a manufacturing environment and is fundamental in the execution of a production plan. Consequently, better coordination and scheduling of production and logistics activities on the shop floor are necessary to survive in the highly competitive environment which requires short lead-time deliveries and low-cost products.

During the past five decades, shop floor scheduling has been a topic intensively addressed in operations research. Although there are numerous methodologies and research studies published in the field of machine scheduling, most of the literature assumes that there is an unlimited number of transporters for delivery of jobs or that transportation times between machines can be neglected, which means that jobs are transported to the next machine immediately. This ideal assumption is not applicable for generating an accurate scheduling on the shop floor. Furthermore, even though transportation times are considered and separated from processing times, most models still assume there are unlimited transporters to move jobs. Integrated scheduling of material handling and manufacturing will involve two types of resources: machines and material handling devices. Either resource could become a bottleneck if not properly scheduled. Thus, considering the transportation issue in the classical machine scheduling will lead to more realistic and practical models. This, in turn, may yield more feasible and accurate production plans for a shop floor. To incorporate these more realistic constraints in this research, not only transportation times are explicitly taken into account but also the availability of transporters is considered.

In Han and McGinnis's study (1989), they estimate that a job spends only 5% of its total cycle time being processed on machines - the job either waits in a queue or is being transported between machines for the remaining flow time. In the study, they conclude that assuming negligible transportation times is not practical for most production systems. Additionally, material handling devices (e.g., robots, automated guided vehicles (AGVs), conveyors, transporters, etc.) are an expensive investment and take a significant portion of the equipment cost in many manufacturing environments. Tompkins and White (1984) indicate the cost of material handling cumulates as high as 80% of the total manufacturing cost of a product. The capital invested on the material handling equipment returns by reducing labor, material or overhead costs, and Meyers and Stephens (2005) indicate the investment should be recovered in two years or less (50 percent return on investment or higher).

In order to reduce the cost and increase the utilization of these automated facilities, the interaction between processing machines and the material handling equipment should be planned carefully. Generally, it is clear that operations in manufacturing and transportation systems must be coordinated carefully in order to achieve ideal overall system performance. To achieve this goal, the following issues regarding transportation scheduling must be addressed simultaneously as suggested by Lee *at el.* (1997):

1. Determine a sequence that specifies the order in which jobs are processed at machining centers;
2. Generate a schedule that makes time-phased routing and dispatching of transporters for job pick-up and delivery; and
3. Design a facility layout and flow paths that make efficient operations possible.

A problem to address the three issues mentioned above at the same time is a combinatorial problem, and it is very difficult to find an optimal solution. Most studies reported in the literature consider at most two of these issues. For example, giving a set of jobs and transporters, the problem is to determine the sequence of jobs processed on the machines and how to transport jobs between machines to minimize the makespan, which is the completion time of the last job on the final machine. In this research, the first two issues, scheduling the job sequence on machines and the transportation plan for jobs, are investigated.

1.2 Research Problems

Two types of flow shop manufacturing environments involving transportation are considered in this research. In the first part of the research, we are interested in a two-machine flow shop scheduling problem with the objective of minimizing makespan. In this flow shop, there are two machines, M_1 and M_2, that are continuously available from time zero for processing n independent jobs j ($j = 1, \ldots, n$) which should pass through M_1 and M_2. All jobs are available at time zero and each machine can handle no more than one job at a time without preemption. In addition, there is a single transporter or material handling device with a given capacity to transport jobs from M_1 to M_2. Transportation times between these two machines are explicitly considered. The problem is to determine the optimal schedule of transporting these jobs to the second machine such that the completion time of the last job finished on the second machine is minimized.

In the second part of the research, a new scheduling problem in an automated manufacturing cell is investigated. The automated manufacturing cell commonly integrates material handling and processing devices to process jobs efficiently. This part considers an application to the manufacturing cell which consists of one loading and unloading station, m processing machines, and a rotary table. The loading/unloading station and the processing machines surround the rotary table. The rotary table is a platform used to carry jobs currently being processed and to transport these jobs to the next processing stations once all jobs on the table complete their current operations. This is shown in Figure 1.1. Jobs have to be loaded on the rotary table at the loading/unloading station, be transported to the processing machines subsequently, and finally be unloaded from the rotary table at the loading/unloading station.

The problem is similar to a flow shop without buffers between machines such that jobs in this flow shop are transported to next machines simultaneously. In addition, unlike the traditional flow shop, jobs will be transported back to the same loading/unloading station for the unloading operation. The characteristic of the re-entrance of jobs increases the complexity of scheduling in this machining center. The objective of the scheduling problem is to determine an optimal job sequence which minimizes the makespan. The manufacturing cell with a single processing machine will be investigated first. Furthermore, an extension which is to increase the number of processing machines in the machining center to two is also considered.

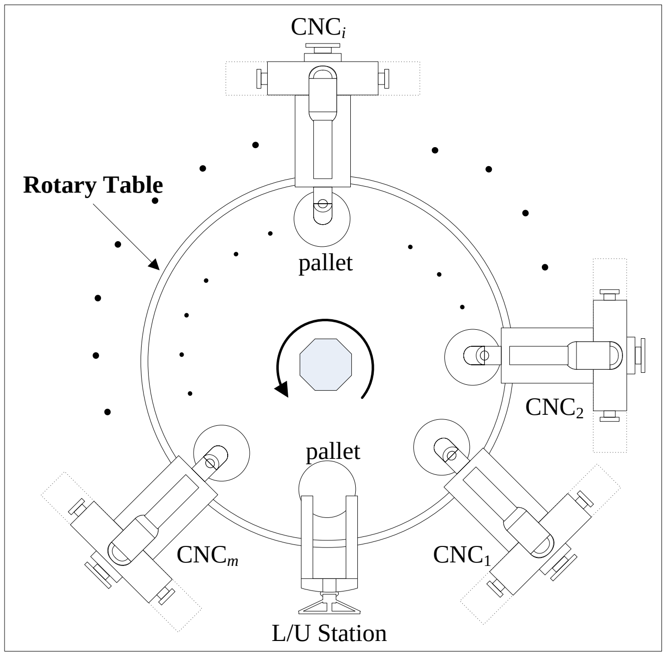


Figure 1.1. A T-line machining center with *m* machines

1.3 Research Objectives and Contributions

For the scheduling problem in a two-machine flow shop with a single transporter, the objective is to develop an improved algorithm to obtain the optimal solution efficiently. The research also explores a new flow shop scheduling problem with two special characteristics: the synchronous job transfer and job re-entrance. For the application in a flow shop, one of the focuses of this research is to identify the complexity and some properties of the problem. In addition, developing algorithms to obtain optimal or near-optimal solutions in a reasonable time is another objective of the research.

Contributions of this study are summarized as follows:

1. For the special case in a two-machine flow shop with transportation considerations, a dynamic programming algorithm is proposed to solve the problem in polynomial time when the transporter's capacity is greater than or equal to a derived threshold value. In an optimal schedule, the maximum number of jobs be transported in every batch is always not greater than the threshold value. The complexity of the algorithm is shown to be $O(n^3)$, where n is the number of jobs.
2. A new flow shop scheduling problem in an automated manufacturing cell with synchronous material movement and job re-entrance is investigated. The complexity of the problem with a single machine is proven to be strongly NP-hard by showing that the problem is equivalent to the numerical matching problem with target sums, which known to be strongly NP-hard.
3. A dynamic programming algorithm is provided to obtain an optimal schedule for the machining center with one machine. The computational effort of the algorithm also shows the exponential time complexity of the problem. However, the dynamic programming algorithm can still obtain an optimal solution efficiently for small and medium size problems. In addition, the dynamic programming algorithm is extended to a two-machine case and a generalized model with m machines.
4. Two-phase heuristic algorithms are developed to obtain an optimal or near-optimal solution for the scheduling problem in a T-line machining center with one machine as well as for the problem with two machines. Two constructive heuristics are proposed to generate an initial sequence for both problems. For the improvement phase, a modified neighborhood search is suggested which integrates the pairwise interchange scheme and a mechanism for avoiding trapping in a local optimum. A sets of computational experiments are conducted including

small, medium, and large-size problems combining three different settings on the loading, processing, and unloading times. The results show that the proposed two-phase algorithms rapidly generate a solution within 1% from the optimum or 3% from the lower bound for the one-machine problem, and 2.3% from the optimum or 6.25% from the lower bound for the two-machine problem.

5. Lower bounds are derived for the one-machine and two-machine problems. The experimental results show that on average the lower bounds from the optimal makespans are at most 2.1% and 4.6% for the small and medium size problems, respectively. Therefore, the proposed lower bounds provide an insightful reference for the optimal value when the optimum is unlikely to be obtained. Furthermore, a method to derive a lower bound is also provided for a generalized model with *m* machines.
6. For the one-machine problem with synchronous material movement, a polynomial time algorithm, $O(n^3 \log n)$, is provided to obtain an optimal sequence when the unloading or the loading times for all jobs are common.

1.4 Thesis Overview

The remainder of this thesis is organized into five additional chapters as follows. Chapter 2 reviews the relevant literature on scheduling, flow shops, job shops, automated manufacturing cells, and complexity theories as well as the research which considers transportation times and transporters.

In Chapter 3, a special case of the two-machine flow shop problem with transportation times is introduced. The problem assumes the processing times on the first machine are common and the capacity of the transporter is greater than or equal to a threshold value. A dynamic programming algorithm is developed to solve the problem. The complexity of the proposed algorithm is analyzed and compared to that of the algorithm developed by Lee and Chen (2001).

Chapter 4 studies the scheduling problem of an automated manufacturing cell consisting of one loading and unloading station, one machine, and one material handling device (a rotary table). There are six main sections in this chapter. First, the problem of sequencing jobs in this manufacturing cell regarding the makespan objective is shown to be strongly NP-hard. Second, a polynomial algorithm is proposed to solve a special case with a constant loading time or unloading time. Third, a dynamic programming algorithm is formulated for this problem and the computational effort of the algorithm is analyzed. Fourth, two constructive heuristics respectively combined with a modified neighborhood search are developed to obtain a high quality solution efficiently for the problem in a large scale. In the last two sections, the experimental designs to evaluate the performance of the heuristic algorithms and the results are presented.

In Chapter 5, an extension of the scheduling problem in the automated manufacturing cell is considered. In this extension, the manufacturing cell also consists of one loading and unloading station and one rotary table, but two machines. The dynamic programming algorithm developed in Chapter 4 is modified for the problem with two machines. Additionally, two heuristic algorithms in the constructive stage are provided to form a sequence as the initial seed for the improvement stage. Similar to the experimental designs for the one-machine problem, the computational evaluation and results for the proposed algorithms are presented. Furthermore, a generalized dynamic programming algorithm and the analysis of its computational effort are presented for the general problem with *m* machines. To obtain a lower bound value for the generalized problem is also presented.

Lastly, conclusion of the study and future research are outlined in Chapter 6.

Chapter 2

Literature Review

2.1 Scheduling

Scheduling is to allocate limited resources to tasks over time such that certain objectives or goals can be achieved or optimized. Leung (2004) states that “Scheduling is a form of decision-making that plays an important role in many disciplines. It is concerned with allocation of scarce resources to activities with the objective of optimizing one or more performance measures”. The resources may take many forms such as number of machines, work force, service points, raw materials, crews and airplanes. The tasks can be manufacturing operations, serving customers, flights of the airplanes, and delivery of goods. The objectives could be the minimization of the completion time of jobs, maximization the number of orders that meet the due dates, or minimization the average service time. Pinedo (1995) also defines scheduling as a decision-making process that exists in most manufacturing and production environments as well as in most information-processing systems. Other examples of scheduling can also be found in transportation and distribution settings as well as in other types of service industries.

In the field of scheduling, there exists a vast amount of research and literature conducted in the past five decades. Many review papers and books in this research area have been published and extensive bibliographies are also available such as Panwalker and Iskander (1977), Graves (1981), Pinedo (1995), Baker (1995), Lee *et al.*(1997), and Framinan *et al.* (2004). Graves (1981) classifies production scheduling problems into three dimensions based on the general characteristics of both scheduling theory and practice: (1) Requirements generation, (2) Processing complexity, and (3) Scheduling criteria. Processing complexity is concerned primarily with the number of processing steps associated with each production task or time. Based on this dimension, production scheduling is commonly categorized as follows:

- One-stage, one-processor (facility)
- One-stage, parallel processors (facilities)
- Multistage, flow shop
- Multistage, job shop

Graves further comments that theoretical insight from simpler problems is often the first step in tackling more complex problems. Hence, one part of this research will consider one-machine case to explore the properties of the problem, and then extend the problem to two machines followed by the generalized problem with *m* machines. Literature regarding flow shop and job shop problems is reviewed in the following sections. Section 2.4 provides a review of literature which considers the transportation times. Section 2.5 reviews the research in the field of scheduling in automated manufacturing cells. In Section 2.6, the complexity theory is presented.

Typically, a scheduling problem is described by a three-field notation $\alpha \mid \beta \mid \gamma$ introduced by Graham *et al.* (1979). The α field describes the machine environment and contains a single entry. Common notations used in the field are summarized as follows:

- F_m : a flow shop with *m* machines.
- J_m : a job shop with *m* machines.
- O_m : an open shop with *m* machines.
- P_m : *m* identical machines in parallel.

The β field provides details of processing characteristics and constraints and may contain no entries, a single entry, or multiple entries. The entries in this field could be preemptions, precedence constraints, machines breakdown, permutation, blocking, recirculation, and particular processing times described as follows:

- Preemptions ($prmp$): Preemptions imply that a job is allowed to interrupt during its processing on a machine at any time.
- Precedence constraints (*prec*): one or more jobs may have to be completed before another job is allowed to start its processing.
- Breakdowns (*brkdwn*): a machine is not available when breakdown occurs.
- Permutation (*prmu*): A constraint that appears in a flow shop requires all machines process the jobs according to the same order.
- Blocking (*block*): Due to a limited buffer in between two successive machines, a job cannot be released to the downstream machine if the buffer is full.
- Recirculation (*recre*): a job may visit a machine more than once.

Because of the new characteristics studied in this research, two notations are introduced in the β field as follows:

- Synchronous material movement ($synmv$): Synchronous material movement implies that jobs are transferred to the next machines simultaneously.
- Reentrance (*re-LU*): a job has to visit the loading/unloading station two times: to be loaded before being processed and be unloaded after completing processes.

The γ field contains the objective to be achieved and usually contains a single entry. The possible objectives could be the makespan, the maximum lateness, the total weight completion time, the total weighted tardiness as well as the weighted number of tardy jobs.

- Makespan ($C_{\max}$): the makespan is the completion time of the last job to leave the system.
- Maximum lateness ($L_{\max}$): the maximum lateness is the worst violation of the due dates among all jobs.
- Total weighted completion time ($\sum w_j C_j$): the sum of the weighted completion times of *n* jobs where w_j is the given weight and C_j is the completion time of job *j*.
- Total weighted tardiness ($\sum w_j T_j$): the sum of the weighted tardiness of *n* jobs where T_j is defined as max(0, C_j - due date of job *j*).
- Weighted number of tardy jobs ($\sum w_j U_j$): the sum of the weighted number of tardy jobs where U_j is equal to 1 when $T_j > 0$; otherwise U_j is zero.

2.2 Flow Shop

Jobs have to be processed on a sequence of machines in the same order, which implies all jobs have identical processing flow. This setting of the manufacturing environment is referred to as a flow shop. In the practical industrial environment, this type of manufacturing is employed due to many advantages it brings for the planning and management of production activities which are enabled by technological developments such as general purpose machines and flexible manufacturing systems (Zegordi *et al.* 1995). In a pure flow shop, there are *m* machines and each job contains *m* operations, each operation requires to be processed by different machines. All jobs are to be processed on every machine in the same flow. As shown in Figure 2.1, based on the definition by Baker (1995), jobs in a general flow shop may require fewer than *m* operations, their operations may not always require adjacent machines in the numbered order. The initial and final operations may not always occur at machine 1 and *m*, but the flow of work is still unidirectional.

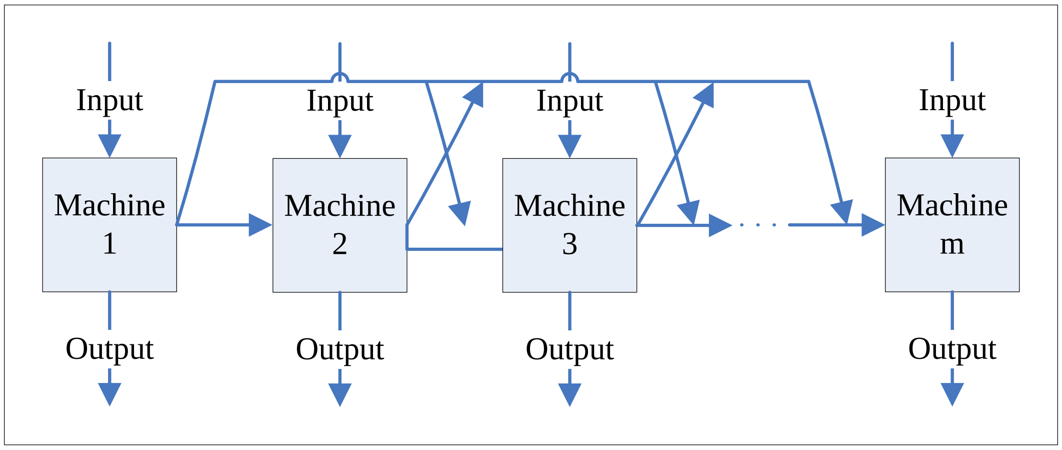


Figure 2.1. Workflow in a general flow shop (Source: Baker, 1995)

The flow shop problem is one of the best known production scheduling problems and the simplest multistage scheduling problem, but it is unfortunately a difficult combinatorial problem. The problem is to determine how to sequence the processing orders on each machine with respect to given criteria. A nonpreemptive schedule with the criterion of minimizing the maximum flow time has raised the most interest in research. The permutation flow shop problem with *n* jobs and *m* machines is commonly defined as follows. Each of *n* jobs is to be sequentially processed on machine 1 to machine *m*. The processing time p_{ji} of job *j* on machine *i* is given. At any time, each machine can process at most one job and each job can be processed on one machine at a time. The sequence in which the jobs are to be processed is the same for each machine. The objective is to find a permutation of jobs that minimizes the makespan. It is well known that for both problems, $F_2 \| C_{\max}$ and $F_3 \| C_{\max}$, there exist optimal solutions that are permutation schedules in which all machines process the jobs according to the same sequence (Conway *et al.* 1967).

The objective of minimizing makespan in a two-machine flow shop problem ($F_2 \| C_{\max}$) can be solved in polynomial time by the well-known Johnson's rule (Johnson 1954): the jobs for which $p_{j1} \le p_{j2}$ are sorted in nondecreasing order of p_{j1} and sequenced first, followed by the remaining jobs sorted in nonincreasing order of p_{j2}. The Johnson's rule has significantly influence on the later research and developments regarding to the flow shop scheduling. Gupta and Stafford (2006) review the major developments in the field of flow shop scheduling in the past five decades since the publication of Johnson's paper. The $F_3 \| C_{\max}$ problem has been shown to be NP-Complete for nonpreemptive schedules by Garey *et al.* (1976). That implies that it is unlikely to find an optimal solution for the problem with $m \ge 3$ in polynomial time. One category of research aims to obtain lower bounds for this type of problems. Lai (1996) presents an $O(mn)$ two-group heuristic algorithm for the *n*-jobs, *m*-machines flow shop permutation scheduling problems, and shows that the algorithm provides $(m+1)/2$ times of the optimal makespan at the worst case. For the problem $F_3 \| C_{\max}$, Chen *et al.* (1996) proposes an $O(n \log n)$ heuristic algorithm based on the Johnson's rule to generate a schedule with makespan at most 5/3 times that of an optimal schedule. On the other hand, a large amount of heuristic algorithms have been developed to yield good approximate solutions to this type of scheduling problems, such as tabu search, simulated annealing, genetic algorithms, ant colony optimization. These heuristics are reviewed and classified in the study by Framinan *et al.* (2004), Hejazi and Saghafian (2005), and Ruiz and Maroto (2005). These studies not only provide an extensive review and evaluation of many heuristics for the permutation flow shop scheduling problem, but also program and test a total of 25 algorithms solving Taillard's (1993) famous 120 instances benchmark.

Metaheuristics

Metaheuristics like Tabu Search, Genetic Algorithms, and Simulated Annealing become common methodologies to solve more complicated and practical flow shop scheduling problems to obtain approximately solution effectively since the prevalence of computer technology. The major literature related to these metaheuristics is discussed below.

Tabu search is particularly designed for escaping from local optimums. This method starts with an initial solution and then applies a move mechanism to search the neighborhood of the current solution to choose the most appropriate one. Ben-Daya and Al-Fawzan (1998) develop a tabu search algorithm, for a flow shop problem with makespan criterion, which generates neighborhoods by the proposed technique and combines a scheme for intensification and diversification that has not been considered before. One of the three methods (swapping, insertion, and block insertion) is randomly selected to generate the next neighbor of the current sequence. Armentano and Ronconi (1999) investigate the application of tabu search to the flow shop scheduling problem for minimizing total tardiness. Grabowski and Wodecki (2004) propose a new very fast local search procedure based on a tabu search approach. In this algorithm, a lower bound on makespan instead of computing the makespan explicitly is used for choosing the best solution to reduce calculations.

Simulated Annealing (SA) is a neighborhood search technique that produces good solutions for combinatorial optimization problems. SA employs certain probability to escape from local optima and the search process can be controlled by the cooling schedule (Hajek 1988). For a flow shop scheduling problem, Zegordi *et al.* (1995) propose an approach which combines the simulated annealing methodology with given specific sequencing information and a tabu search feature. Tina *et al.* (1999) focus on the generation mechanism of the simulated annealing algorithm, and six types of perturbation schemes for generating random permutation solutions are introduced. They demonstrate that the SA algorithm can produce very efficient solutions to different combinatorial optimization problems by adopting a proper perturbation scheme. In order to enhance the performance of the genetic search and to avoid premature convergence, Wang and Zheng (2003) propose a hybrid heuristic which replaces the mutation operator by the SA's metropolis sample process. The metropolis sample process replaces the mutation operator with a mutation rate adjusted by the controlled temperature to control the search behavior. Low (2005) addresses a flow shop scheduling problem with unrelated parallel machines by a simulated annealing-based heuristic. This problem considers independent setup times as well as dependent unloading times, and the objective is to minimize the total flow time.

Genetic algorithms (GAs) are powerful search techniques which have been widely applied to many optimization fields. The concepts of genetic algorithms are based on the mechanics of natural selection and natural genetics. The paper by Reeves (1995) attempts to apply GAs to the flow shop sequencing problem. This research shows that GA will perform relatively better for large-size problems, reach a near-optimal solution rather more quickly, compared to simulated annealing algorithms. Ponnambalam *et al.* (2001) develop a genetic algorithm and compare it with five heuristics for the makespan objective to solve the flow shop problems. The proposed genetic algorithm is found to yield much better quality solutions and computationally efficient as well. Wang *et al.* (2003) propose an order-based genetic algorithm which is inspired by ordinal optimization to ensure the quality of the solution found. They show that a good enough solution can be guaranteed with a high confidence level and reduced computation effort by numerical simulation results.

The ant colony system (ACS) first proposed by Dorigo and Gambardella (1997) is one of the most recent and promising metaheuristics for combinatorial optimization problems. Ant colony optimization (ACO) simulates the collective foraging habits of ants, venturing out for food and bringing it back to the nest (Hejazi and Saghafian 2005). Ying and Liao (2004) develop an ACS algorithm for a permutation flow shop scheduling problem with minimizing makespan as the objective. The proposed algorithm is compared with other metaheuristics such as genetic algorithm, simulated annealing, and neighborhood search from the literature. The computational results show that the ACS algorithm is a more effective metaheuristic. Rajendran and Ziegler (2004) consider the objective of minimizing total flowtime in a flow shop scheduling problem and propose ACS algorithms. The proposed algorithms have been applied to 90 benchmark problems taken from Taillard and yield better solutions compared with the other heuristics.

Stochastic Flow Shop Scheduling

Most of the scheduling models that have been developed assume that parameters of problems are deterministic. Regarding the development of stochastic models in flow shop problems, Gupta and Stafford (2006) indicate research that includes stochastic assumptions is not prevalent. However, stochastic scheduling problems still attract a substantial number of researchers to work on them.

In a two-machine flow shop, when the processing times on both machines are independent and exponential random variables, Talwar (1967) developed a rule to sequence jobs so that the sequence minimizes the expected makespan. Pinedo (1983) consider one-machine scheduling problem in which the job processing times are independent exponentially distributed random variables. They show that simple policies often minimize such criteria as the expected weighted sum of completion times and weighted number of late jobs. A flow shop scheduling problem with *m* identical machines in which the job processing times are random variables is investigated in the paper by Pinedo (1985). Sequencing jobs in the descending order of expected amount of processing times stochastically minimizes the makespan. Kijima *et al.* (1990) also study a stochastic flow shop scheduling problem with *m* identical machines. In this flow shop, however, the buffer sizes between machines are not unlimited. They provide a schedule policy which generates the minimum expected makespan.

Allahverdi and Mittenthal (1995) consider a two-machine flow shop problem with stochastic machine breakdowns. Under certain conditions on the distributions of the machine breakdowns, the Johnson's rule minimizes the makespan stochastically. Kamburowski (1999) consider a two-machine stochastic flow shop with an unlimited storage between machines. They present a sufficient condition which has less restrictive assumptions on the job processing time distributions for the optimal schedule when the objective is to minimize the makespan.

Balasubramanian and Grossmann (2002) consider a flow shop scheduling problem with uncertain processing times described by discrete probability distributions. They propose a branch-and-bound algorithm based on a probability disaggregation scheme. The value obtained by letting the uncertain processing times be replaced with their mean values is the lower bound on the expected makespan for a given sequence. They also show the algorithm provides excellent approximations to the expected makespan of a given sequence for the case of continuous probability distributions of certain forms by using a discretization scheme.

Gourgand *et al.* (2003) develop a recursive algorithm to evaluate the performance of the *m*-machine flow shop scheduling problem with exponentially distributed job processing times. The algorithm is based on Markov chains to compute the expected makespan and a discrete event simulation model to evaluate the expected makespan. Several heuristics (e.g., Rapid Acess) and metaheuristics (e.g., simulated annealing) are integrated with the recursive algorithm to obtain near-optimal solutions in a short time for two-machine problems. Wang and Zhang (2006) propose a simulated annealing approach combined with hypothesis test for an *m*-machine flow shop scheduling problem. In this flow shop setting, the job processing time is a random variable with uniform distribution. By using hypothesis test, solution performance can be reasonably estimated so that the searching efficiency can be improved.

Kalczynski and Kamburowski (2006) assume job processing times are independently and Weibull distributed random variables with a common coefficient of variation. They propose a sequencing rule for the expected makespan minimization. The simulation experiments indicated that the rule might find a schedule with the minimum expected makespan, but its optimality cannot be proven analytically.

2.3 Job Shop

In a job shop, each job can have its own routing which is the sequence of being processed on machines, but these routings are predefined and fixed. In a flow shop, there is a single routing; that is all jobs are sequentially processed by machines. In a job shop, however, each type of jobs could pass through a set of machines in different sequences as shown in Figure 2.2. Job shop scheduling is to find the job sequences on each machine based on an objective while a job's routing is given. It is the most general production scheduling problem and it seems to be capable of capturing the nature of most production environments. In the two-machine problem, for example, some of the jobs are processed from machine 1 through machine 2, but the others go through machine 2 first and then machine 1. Additionally, there could be some jobs only requiring one of the machines. The problem $J_2 || C_{\max}$ can be reduced to or equivalent to $F_2 || C_{\max}$ and be solved by the Jackson's rule.

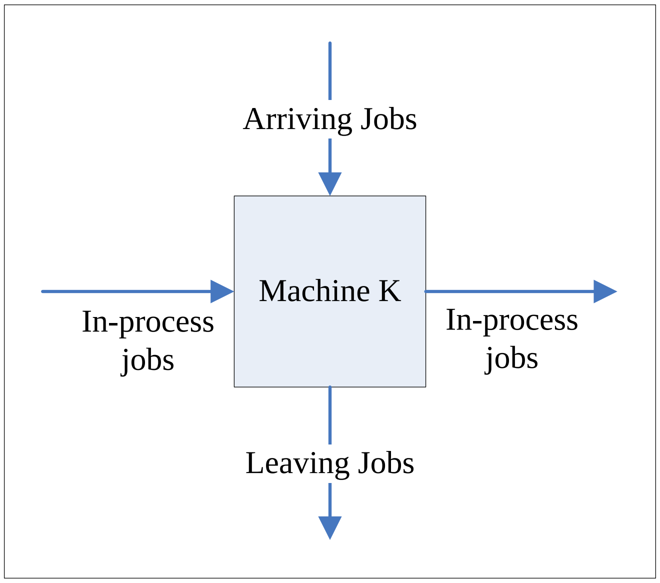


Figure 2.2. Workflow in a job shop (Source: Baker, 1995)

Although the job shop is more flexible and has less restriction on jobs, it is the most difficult production scheduling problem to solve from the perspective of optimization. It is well known that most of job shop scheduling problems are NP-hard (Lenstra and Rinnooy 1979). Hence, nonpreemption of jobs and minimizing makespan are the most common assumptions. Graves (1981) indicates that branch and bound is the most common optimization approach to address the job shop problem where various procedures differ primarily with respect to the branching rules, the bound mechanism, and the generation of bounds. In addition, the problem consisting of ten jobs and ten machines is the well-known benchmark introduced by Fisher and Thompson in 1963. Even for a small-size problem, it is difficult to find optimal scheduling efficiently, not to mention a large-scale problem. In the paper by Jain and Meeran (1999), a broad review on the state-of-the-art job shop scheduling techniques are provided.

Pezzella and Merelli (2000) propose a heuristic method for solving the minimum makespan problem of job shop scheduling. The proposed local search method is based on a tabu search technique and a shifting bottleneck procedure to generate the initial solution and refine the next current solutions. Ponnambalam *et al.* (2000) develop a tabu search algorithm for job shop scheduling problems and adopt an adjacent pairwise interchange method to generate the neighborhoods. Hurink and Knust (2002) consider a single transporter scheduling in a job shop environment. The objective is to determine a sequence to minimize the sum of all traveling and waiting times of the transporter. They present a tabu search algorithm for this problem and show that the algorithm yields a good upper bound in a short amount of time. Nowicki and Smutnicki (2005) provide a tabu search-based algorithm which adopts some elements of path relinking techniques to generate initial solutions. The computational results show that the proposed algorithm offers a very accurate solution to solve the job shop problem with the makespan criterion in a short time.

Wang and Zheng (2001) develop a general, parallel, and easily implemented hybrid optimization framework, and apply it to job shop scheduling problems by combining two global probabilistic search algorithms: GA and SA. During the hybrid search process, GA provides a set of initial solutions for SA at each temperature to generate neighbor solutions, and GA uses the solutions found by SA to continue parallel evolution. Dai and Weiss (2002) propose a heuristic algorithm which uses safety stock and keeps the bottleneck machine busy at most of the time, while the other machines are constrained by the bottleneck machine. Kis (2003) develops two heuristic algorithms: a tabu search and a genetic algorithm for a job shop scheduling with alternative routings. They demonstrate that the tabu search is superior to the GA both in terms of solution quality and computation time. Mattfeld and Bierwirth (2004) consider job shop scheduling problems with release and due-dates, and with various tardiness objectives. They employ the GA with a heuristic reduction of the search space which helps the algorithm to find better solutions in a shorter computation time. Two ways of reducing a search space are investigated by considering decisions made at machine and shop floor level. Gonçalves *et al.* (2005) propose a hybrid genetic algorithm for the job shop scheduling problem. The schedules are constructed using a priority rule in which the priorities are defined by the genetic algorithm. After a schedule is obtained, a local search heuristic is applied to improve the solution. The algorithm produces solutions with an average relative deviation of 0.39% to the best known solution on a set of 43 testing problems.

2.4 Flow Shop with Transfer Times and Transporters

Extensive literature can be found in machine scheduling involving time lag which is the time between the completion of an operation and the beginning of the next operation of a job in a production system. It can be referred to as the transportation, cooling, or heating time. In our research, the time lag is considered as the transportation time which is attributed to the actual transportation of a job between the processing machines by transporters or AGVs. In the classical models, it is assumed that jobs can be transported between machines instantaneously. The ideal assumption would not be applicable to most practical production environments. There are two types of transportation time consideration in the literature: one considers only the time lag, which implies transporters are always available such as Szwarc (1983), Dell'Aimco (1996), Schutten (1998), Strusevich (1999), Rebainel and Strusevich (1999), and Karuno and Nagamochi (2003); the other explicitly takes both transportation time and availability of transporters into consideration such as Levner *et al.* (1995), Hurink and Knust (2001), Oulamara and Soukhal

(2001), Lee and Chen (2001), Lee and Strusevich (2005), and Soukhal *et al.* (2005). The details of these papers are described later in this section.

In these models, several attributes can be configured according to real manufacturing systems: (1) processing times on machines, (2) transportation times between machines, (3) number of transporters, and (4) capacity of a transporter. Both processing times and transportation times for jobs can be characterized as job-independent and job-dependent. The number of transporters and its capacity could be greater than one to increase the complexity and practicality of a model.

The job-dependent processing times can be found in production lines with mix jobs or products where different types of jobs require different processing times. The two-machine flow shop problem with constant transportation times but job-dependent processing times has been shown to be NP-hard in the strong sense by Hurink and Knust (2001). The job-independent transportation times can be found in the production processes when the transportation times depend on the distance between the machines, rather than on the weight or size of a job to be transported. The job-dependent transportation times, on the other hand, would consider attributes of jobs such that each job has different transportation times. Most research assumes job-independent transportation times because the problem with job-dependent transportation times is proven to be strongly NP-hard (Yu 1996; Hurink and Knust 2001). Moreover, the times of a transporter to travel back and forth between two machines can be another variable of the model. If these two travel times are job-independent but not equal, the problem has also been shown as strongly NP-hard (Yu 1996; Yu *et al.* 2004).

When physical transporters or AGVs are considered in transportation models, only one transporting device is assumed in most studies. Lee and Chen (2001) not only consider multi-transporters in the two-machine flow shop problem but also develop a dynamic programming algorithm to tackle the problem in a special case where identical processing times on one of the two machines is assumed. Furthermore, the question regarding to capacity of a transporting device will arise if the device is physically incorporated in the model. Carrying one job at a time is the most common assumption and simpler to address. In these study (Lee and Chen 2001; Oulamara and Soukhal 2001; Lee and Strusevich 2005; Soukhal *et al.* 2005), several cases of the capacity of a transporter greater than one are studied. In our study, the capacity of a transporter will also be considered.

In the following paragraphs, the literature which is relevant to our research is elaborated: Szwarc (1983) considers two models for flow shop problems: one with time lags between machines and another one with distinct setup, processing, and release times. Lower bounds of the completion time for both cases are developed by solving some classical two-machine flow shop problems, and choosing the best permutation as an approximate solution. Levner *et al.* (1995) consider a flexible manufacturing cell consisting of two machines, automated storage/retrieval stations, and one transporting robot. Transportation between the input/output stations and machines, and between two machines is performed by a mobile transporting robot. Loading and unloading operations are also performed by the same robot. The transporting robot can carry only one job at a time, and there is no buffer storage for work-in-process (WIP). They solve the problem in polynomial time by using a graph-based approach for a small-scale flexible manufacturing cell with job-dependent processing and material-handling operations.

Lee *et al.* (1997) review several articles which consider transportation time. The general model of the problem assumes all jobs are ready at the time zero and each job has its own routing and processing time. There are identical transporters to deliver jobs between machines, and these transporters travel on a shared network where no traffic collision can occur. All the operations by the transporters are non-instantaneous and non-preemptive. Neither a machine nor a transporter can hold more than one job at any time. The problem is to find a schedule for job sequencing and time-phased dispatching and routing of transporters so that a given objective is optimized. They divide recent work related to this model into three categories:

(1) Robotic cell scheduling;

(2) Scheduling of automated guided vehicles; and

(3) Cyclic scheduling of hoists subject to time-window constraints.

In the paper, they also review some of recent developed methodologies for scheduling problems on these topics. The general versions of these problems are all NP-hard in the strong sense. Most of these problems, especially those encountered in real systems, are so complicated that a formal mathematical formulation can’t be obtained.

Anwar and Nagi (1998) indicate the basic elements of AGV system design: (1) guide path network design (2) optimal number of AGVs, (3) vehicle dispatching, (4) vehicle routing and (5) traffic control. They consider simultaneous scheduling of material handling transporters and manufacturing equipments (such as machines and workcenters) in the production of complex assembled products and propose an effective heuristic algorithm which employs a critical-path-based scheduling approach and a just-in-time methodology to minimize the production makespan of large and complex assemblies as well as WIP costs.

In the paper by Schutten (1998), transportation times are taken into consideration by extending the Shifting Bottleneck (SB) procedure for the classical job shop to deal with practical features. The SB procedure decomposes the problem of scheduling a classical job shop into a series of single-machine scheduling subproblems such that these practical features can be easily applied. In addition, the SB procedure generally produces good solutions for job shop problems in relatively short computation time compared to tabu search and simulated annealing algorithms.

Strusevich (1999) proposes a heuristic algorithm to solve two-machine open-shop scheduling problem with arbitrary transportation times in polynomial time. In an open shop problem, the processing routings of jobs are not given in advance, but have to be determined with respect to the objective criteria. The algorithm is an extension from the Gonzalez-Sahni algorithm which partitions the set of jobs into two subsets and finds the permutation that defines a certain flow shop schedule for each subset. The analysis of the algorithm shows that the resulting makespan is at most 3/2 times the optimal value at any worse case. Rebainel and Strusevich (1999) consider the problem with special transportation times. If the largest transportation time does not exceed the smallest processing time, the optimal schedule can be obtained by the proposed linear time algorithm. They also present an algorithm that creates a heuristic solution to the problem with job-independent transportation times and show that the algorithm provides a worst-case performance ratio of 8/5 if the transportation time of a job depends on the assigned processing route. The ratio reduces to 3/2 if all transportation times are equal. Given that *n* is the number of jobs, Karuno and Nagamochi (2003) prove that the bound can be improved by designing an algorithm that delivers a (11/6)-approximation solution in $O(n \log n)$ time.

Another type of problem consists of two machines and one transporting robot. Processing times of jobs are arbitrary, but transportation times are constant for all jobs. The problem was shown to be NP-hard in the strong sense by Hurink and Knust (2001). Jobs have to be transported by the robot between machines, and only one job can be carried at a time. Unlimited buffer space is assumed. Additionally, they also show the problem with constant processing times on both machines, but arbitrary transportation times between machines for each job is strongly NP-hard. The similar proof can also be found in Yu's research (1996; Yu *et al.* 2004) which assumes the constant processing time equal to one. In a special case, a polynomial algorithm is proposed to solve the multi-machines flow shop problem with all processing times equal to one and transportation times between two adjacent machines are constant.

In the paper (Lee and Chen 2001), transportation time is explicitly considered. In addition, multiple transporters and capacity are also considered. They study two types of transportation problems. The first one, denoted as "Type-1", considers intermediate transportation in a flow shop between two machines. The second one, denoted as "Type-2", is to analyze the delivery of finished jobs to customers. The computational complexity of different scenarios is shown and open problems are also underlined. One of these problems with two machines, one transporter, and the capacity of the transporter greater than three, is shown to be strongly NP-hard even if the transportation times are all equal. If processing times of all jobs on first machine or on the second machine are equal, the problem is solvable by a dynamic programming algorithm proposed in this paper even when there are multiple transporters with capacity greater than 1. Lee and Strusevich (2005) present the best possible (3/2)-approximation algorithm with at most two shipments for the two-machine flow shop problem with one unlimited capacity transporter.

The paper (Oulamara and Soukhal 2001) investigates flow shop scheduling models that explicitly consider constraints on both transportation times and buffer capacities with the objective function of minimizing the makespan. Finished jobs also need to be delivered to a customer or a warehouse by one vehicle which is assumed to have capacity of two jobs. The problem could be regarded as the “Type-2” problem according to Lee and Chen (2001). Based on these assumptions, they show the problem with unlimited buffer between machines is strongly NP-hard as well as the problem with no buffer storage between machines. Due to the complexity of the problems, they propose four greedy algorithms to solve the problems and compare the performance of these methods. Furthermore, Soukhal *et al.* (2005) prove that the Type-2 problems with the vehicle’s capacity equal to three are NP-hard in the strong sense.

2.5 Automated Manufacturing Cell

Most of the literature regarding automated machining centers or robotic cells considers the scheduling of jobs and robot moves between machines. In these manufacturing cells, parts are usually loaded and unloaded in different locations and material movement between stations is asynchronous. Sethi *et al.* (1992) study the problem of sequencing jobs and robot moves in a robotic cell where a single robot is used to transport jobs between stations. The cell is a flow shop system where jobs pass sequentially through the input station, machine stations, and the output station. They show that only two possible optimal policies of robot moves exist for the two-machine robotic cell scheduling problem with a single part type. For the problem with multiple part-types, a polynomial time algorithm is derived to minimize cycle time for a given fixed sequence of robot moves.

Levner *et al.* (1995) propose a polynomial-time algorithm to obtain the minimum makespan for a two-machine robotic cell. In this robotic cell, there are two robots dedicated to load and unload jobs in each machine, and the loading and unloading times are job-dependent. There is also a transporting robot to move jobs from the first machine to the second machine. In addition, a job completed on the first machine should be transported to a storage buffer which is located in the range of the robot dedicated for the second machine.

Logendran and Srisjandarajah (1996) develop analytical methods for determining an optimal sequence of jobs and robot moves with minimum cycle time in three different types of two-machine robotic cells: a robot-centered cell, a mobile robot cell, and an in-line robot cell as shown in Figure 2.3, Figure 2.4, and Figure 2.5. They consider the scheduling of single part-type and multiple part-types problems in these three cellular layouts, respectively.

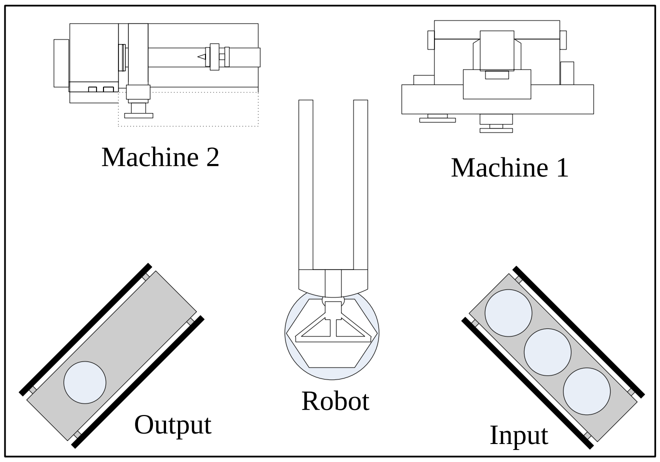


Figure 2.3. Robot-centered cell layout (Logendran and Srisjandarajah 1996)

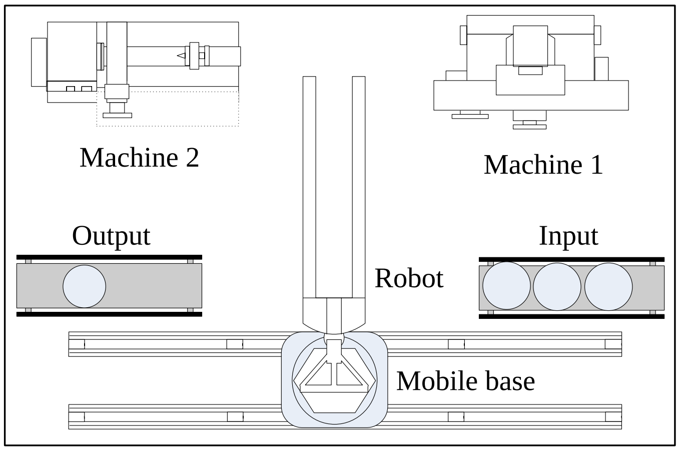


Figure 2.4. Mobile robot cell layout (Logendran and Srisjandarajah 1996)

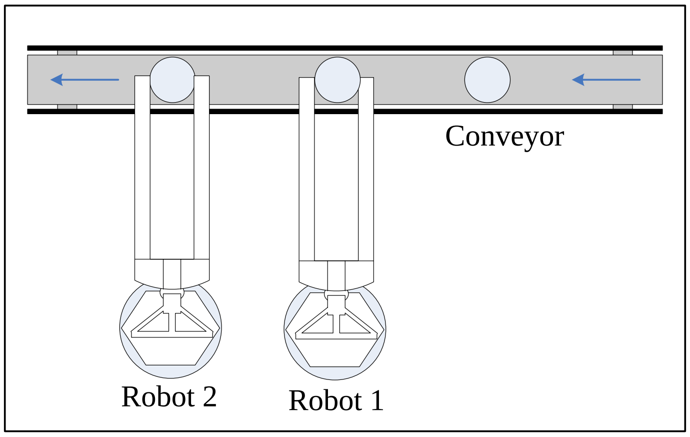


Figure 2.5. In-line robot cell layout (Logendran and Srisjandarajah 1996)

Given that n is the number of jobs, Hall *et al.* (1997) provide a $O(n^4)$ time algorithm to obtain an optimal part sequence and robot moves in a two-machine robotic cell with multiple part-types. Aneja and Kamoun (1999) formulate this problem as a traveling salesman problem with a special cost structure, and improve the complexity of the algorithm from $O(n^4)$ to $O(n \log n)$. Dawande at el. (2005) present a survey and summary of the recent developments regarding scheduling in robotic cells. They provide a classification scheme for the scheduling problems of robotic cells based on the characteristics of the manufacturing cells such as robot devices, machine environments, and processing restrictions. They also discuss implementation issues and the use of optimal policies for different system settings.

Synchronous transportation of jobs between stations is a particular characteristic of the machining center addressed in this study. Without considering the L/U station, if there are only two machines or stations in the machining center with the mechanism of synchronous material movement, the problem is equivalent to a two-machine flow shop problem with blocking which can be solved in polynomial time by Gilmore-Gomory algorithm (Gilmore and Gomory 1964). Hall and Sriskandarajah (1996) prove a three-machine flow shop problem with blocking is strongly NP-complete. However, a three-machine flow shop with synchronous material movement is not reducible to the problem with blocking, because the constraint of blocking only restricts transfer of a job between two successive machines. In the problem with blocking, for example, a job completed on the second machine can be released to the third machine while it becomes idle. In the case of synchronous transfer, however, the job processed by the second machine can only be released to the downstream machine while both of the first and third machines finish their current operations.

Soylu *et al.* (2007) consider a flow shop scheduling problem with synchronous transfer between stations. They develop a brand-and-bound algorithm with several lower and upper bounds to efficiently obtain the minimum makespan for a moderate-sized problem. They indicate this type of manufacturing system with synchronous transfer is advantageous when set-ups for a transporter are timely or costly, or when buffer spaces are limited between stations or jobs are physically large.

2.6 Computational Complexity Theory

The notion of complexity refers to the computing effort required by an algorithm to solve a problem. The computing effort could be measured by time and space required by a computer. Running time of an algorithm can also be measured by the number of steps or elementary operations required. The goal of the complexity theory is to classify problems and algorithms according to the computing effort required. Before developing or constructing the solution to a given problem, the knowledge about the complexity of the problem will provide valuable information to decide what approaches are more suitable to be employed.

There are two classes to categorize a problem: the class ***P*** and ***NP***. A problem belongs to the class ***P*** is that there exists an algorithm to solve the problem in polynomial time. A polynomially solvable problem requires computing effort bounded by a polynomial in the length of the problem encoding. The class ***NP*** is essentially the set of problems which do not have polynomial time algorithms. A problem is called NP-hard if every problem in the class ***NP*** polynomially reduces to that problem. Not all problems within NP-hard class are equally difficulty. A problem, referred to as strongly NP-hard, can not be solved in a polynomial time even as a function of the size of the problem in unary encoding. Otherwise, the problem is referred to as NP-hard in the ordinary sense or simply NP-hard. Furthermore, if an optimization problem is NP-hard, the associated decision problem is referred to as NP-complete.

According to the paper by Garey and Johnson (1979), two properties of the NP-complete problems can be concluded as follows:

(1)No NP-complete problem is known to be solvable by a polynomial time algorithm.

(2)If a polynomial time algorithm for one of the NP-complete problems exists, polynomial time algorithms will be obtained for all the NP-complete problems.

As a result, once a problem can be proven to be NP-complete, it is wise to adopt heuristic approaches to obtain an approximate solution because the existence of a polynomial time algorithm to obtain the optimal solution is unlikely. During the past several decades, many problems have been shown to be NP-complete. The principal technique to demonstrate NP-completeness of a given problem is to transform any instances to a known NP-complete problem. If any instance of the given problem is polynomially reducible to a NP-complete problem and the transformation also holds in the opposite direction, then the given problem can be categorized as NP-complete. One of the well-known problems is the 3-partition (Garey and Johnson 1979), which is commonly adopted to prove a given problem as NP-complete in the strong sense. Yu (1996), Lee and Chen (2001), and Hurink and Knust (2001) prove different configurations of flow shop problems with transportation times as NP-complete by transforming them as 3-Partition problem presented as follows.

3-Partition Problem

Instance: Given a set of A which includes 3m elements: $A = \{a_1, a_2, ..., a_{3m}\}$ .Each element in A is an integer number satisfying $\frac{B}{4} < a_i < \frac{B}{2}$;$i = 1, ..., 3m$ and such that $\sum_{i=1}^{3m} a_i = mB$ for some integer *B*.

Question: Does there exist a partition of the set *A* into *m* disjoint subsets with 3 elements $A_1, A_2, ..., A_m$ such that $\sum_{i \in Aj} a_i = B$ for $j = 1, ..., m$?

In scheduling problems, the complexity of problems with different settings has been investigated and studied intensively. Garey and Johnson (1979) provide an extensive list of known NP-complete or NP-hard problems for reference. Furthermore, a hierarchy of problems that describes the relationships between hundreds of scheduling problems is established due to considerable devotion of researchers. The hierarchy could provide information on how the complexity of a problem will be affected when one of settings or objectives is changed. Figure 2.6 shows the complexity hierarchies of deterministic scheduling problems associated with machine environments and objective functions.

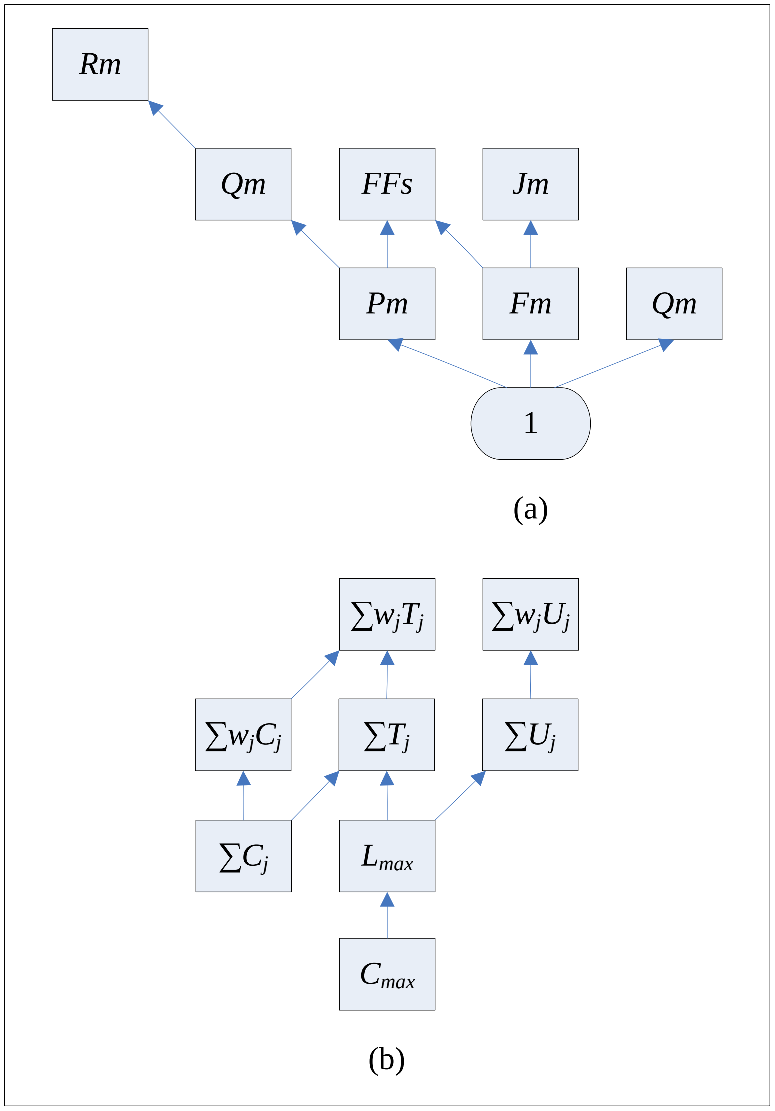


Figure 2.6. Complexity hierarchies of deterministic scheduling problems: (a)machine environment, (b) objective functions (Source: Pinedo, 1995)

Chapter 3

Two-machine Flow Shop with Transportation Considerations

3.1 Introduction

In a flow shop manufacturing system, semi-finished jobs are transferred from one machine to another by transporters. Each job has to be sequentially processed on machine 1 first, then machine 2, and until the last machine *m*. Typically, transportation times between machines are neglected or the availability of transporters are ignored. In this study, however, not only the transportation times but also the availability of transporters are explicitly incorporated into the model. As this production system, a two-machine flow shop model is considered.

In the two-machine flow shop, all jobs start on machine 1 and finish operations on machine 2. Each machine can only process one job at a time, and preemption is not allowed. All jobs are available at time zero and wait for processing in the input buffer of machine 1. The processing time on machine l for job j is denoted as p_{j1} and on machine 2 as p_{j2}. After the operation on machine 1 is completed, jobs are stored at the output buffer of machine 1 and wait to be transported to machine 2. Jobs transported together in one shipment from machine 1 to machine 2 by a transporter are defined as a batch. After being transported to machine 2, these jobs wait to be processed in the input buffer of machine 2. The buffer sizes are assumed to be unlimited. The transporter takes t_1 to travel from machine 1 to machine 2, and t_2 to return to machine 1. The departure time of k^{th} batch on machine 1 is denoted as d_k. We also assume that there is one transporter in the system and its capacity is denoted as *c*. Loading and unloading times of jobs on machines are negligible or they are assumed to be included in processing times. Similarly, times to load and unload jobs on the transporter are neglected or they are assumed to be included in the transportation times.

The performance measure is makespan, *Cmax*, which is widely adopted as an objective criterion because of its simplicity for mathematical formulation and theoretical analysis. In the study, we also set the minimum makespan as the objective and follow the commonly used three-field notation $\alpha | \beta | \gamma$ to represent a machine scheduling problem. In the α field, TF_2 denotes the two-machine flow shop scheduling problem with transportation times between machines which is used in the paper by Lee and Chen (2001). In the β field, *v* denotes the number of transporters, and *c* denotes the capacity of the transporter. In the r field, $C_{\max}$ is the objective of the problem. Hence, the scheduling problem to minimize makespan in a two-machine flow shop with *x* transporters and the capacity of each transporter equal to *y* is represented as $TF_2 | v = x, c = y | C_{\max}$. The notations for the problem are summarized as follows:

- *n*: total number of jobs need to be scheduled,
- *j*: index of a job, *j*=1, 2, …, *n*,
- *i*: index of a machine, *i*=1, 2 ,
- p_{ji}: processing time of job *j* on machine *i*,
- *v*: number of transporters,
- *c*: capacity of a transporter (number of jobs that can be carried at a time) ,
- *u*: maximum number of jobs to be transported in each batch ($u \leq c$),
- $C_{\max}$: the completion time of last job,

- t_1:busy transportation time from machine 1 to machine 2,
- t_2: empty transportation time from machine 2 to machine 1,
- d_k : the departure time of k^{th} batch on machine 1.

3.2 Dynamic Programming Algorithm for $TF_2 \mid p_{j1} \equiv p_1, v = 1, c \geq u \mid C_{\max}$

In general, the two-machine flow shop problem could be decomposed as two subproblems. The first subproblem is to determine a sequence of jobs on machine 1. The second subproblem is to develop a schedule of departure time points for transporters on machine 1. According to the paper by Hurink and Knust (2001), the two-machine flow shop problem with one transporter and its capacity is one ($TF_2 \mid v = 1, c = 1 \mid C_{\max}$) is strongly NP-hard. However, Lee and Chen (2001) have shown that if the following assumption 3.1 is added, then the sequence of jobs on machines can be predetermined, thus the problem becomes polynomially solvable.

Assumption 3.1. The processing times on machine 1 for all jobs are job-independent, namely, the processing times are equal to a constant denoted as $p_{j1} \equiv p_1$ for all j .

Under Assumption 3.1, the jobs are sequenced in the non-increasing order of p_{j2} (the longest processing time first) on both machines such that there exists an optimal permutation sequence. Furthermore, if the processing times on machine 2 for all jobs are constant, the jobs will be sequenced in the non-decreasing order of p_{j1} (the shortest processing time first) on both machines. In both cases, the scheduling problem is simplified and only a departure schedule of jobs has to be determined given the optimal job sequence. Therefore, Lee and Chen (2001) propose a dynamic programming algorithm to solve the problem $TF_2 \mid p_{j1} \equiv p_1, v \geq 1, c \geq 1 \mid C_{\max}$ optimally in polynomial time. In this research, an improved dynamic programming algorithm is proposed when there is one transporter and the capacity of the transporter is greater or equal to a threshold value. In this special case, the number of jobs in a batch transported to machine 2 is always less than or equal to this threshold value in an optimal schedule. The threshold value will be derived later.

Lee and Chen (2001) have proven several properties that hold for two-machine flow shop problems with transportation. Those properties are also necessary conditions for deriving our algorithm.

Property 3.1.(Lee and Chen 2001) There exists an optimal schedule for the problem $TF_2 \mid v \geq 1, c \geq 1 \mid C_{\max}$ problem that satisfies the following conditions.

(i) Jobs are processed on machine 1 without idle time.

(ii) Jobs transported in the same batch are processed consecutively without idle time on both machines.

(iii) Jobs finished earlier on machine 1 are delivered earlier to machine 2. Furthermore, the sequence of jobs on machine 1 is the same as that on machine 2. Namely, it is a permutation schedule.

(iv) The departure times of two consecutive batches delivered satisfy that either $d_{k+1} = d_k + t$ or d_{k+1} is the completion time of the last job in the $k+1^{th}$ batch on machine 1, where $t = t_1 + t_2$ is the transportation time of a round trip to travel between machine 1 and machine 2. When d_{k+1} is equal to the completion time of the last job in the $k+1^{th}$ batch on machine 1, d_{k+1} is referred to as an integer departure point; otherwise, it is called as the immediate departure point.

In addition, a property regarding the threshold value of the transporter's capacity can be derived, while the processing times for all jobs on machine 1 are identical (Assumption 3.1).

Property 3.2. The threshold of the transporter's capacity (u) is $\left\lceil \frac{2(t_1+t_2)}{p_1} \right\rceil - 1$ given $p_{j1} \equiv p_1$.

Proof. Assume the transporter's capacity is greater than u, and there is a batch, say batch i, containing u+1 jobs. Batch i is transported to machine 2 by the transporter at the departure point d_i, which is the time for the transporter leaving from machine 1. Thus, the arrival time at machine 2 of the transport is $d_i + t_1$. The total processing time for batch i on machine 1 is equal to $(u+1) * p_1$ or $\left\lceil \frac{2(t_1+t_2)}{p_1} \right\rceil * p_1$ which is greater than or equal to the travel time of two round trips ($2(t_1+t_2)$). Hence, batch i can be split into two small batches: $\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$ and $\left\lceil \frac{t_1+t_2}{p_1} \right\rceil$. Assume that these two batches will be transported to machine 2 as two shipments and the second batch will also be transported at the departure point d_i. In order to ensure the second batch transported at d_i, the transporter has to deliver the first batch at the departure point d_i^1 which cannot be later than d_i minuses the time of one round trip ($d_i^1 \le d_i - (t_1+t_2)$) as shown in Figure 3.1.

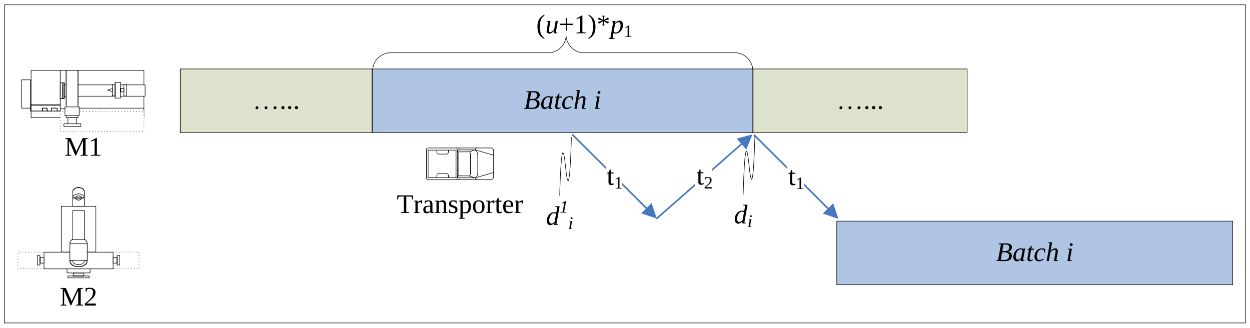


Figure 3.1. Batch *i* containing *u*+1 jobs transported to machine 2 at d_i

Let $d_i^1 = d_i - (t_1 + t_2)$. Because the total processing time for jobs in the first batch, $\left\lfloor \frac{t_1 + t_2}{p_1} \right\rfloor p_1$, is less than or equal to $t_1 + t_2$, the first batch are ready for shipping before d_i^1. In additional, the returning time of the transporter is also prior to d_i^1 since $(u+1)p_1 \geq 2(t_1 + t_2)$. Hence, the transporter can return back to machine 1 prior to d_i after transferring the first batch to machine 2. Thus, the arrival time of the second batch on machine 2 is identical as the arrival time of transporting these jobs in one batch. Figure 3.2 illustrates the delivery of these two batches.

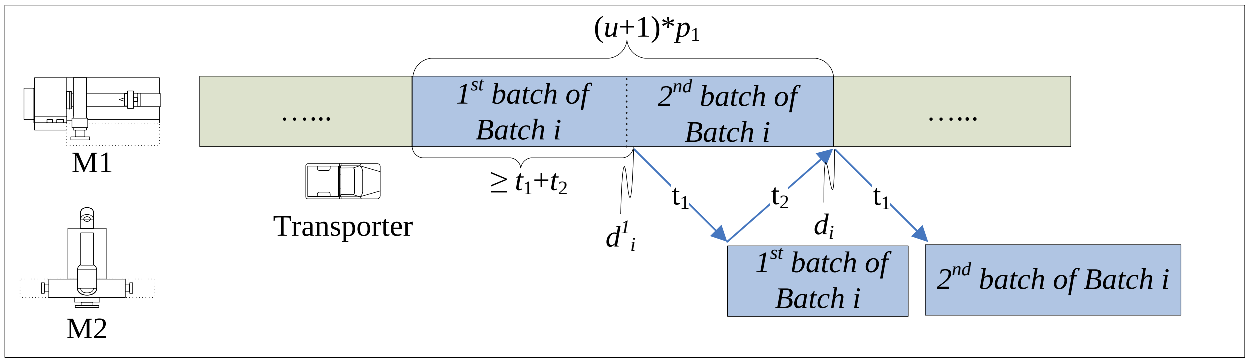


Figure 3.2. Batch *i* is split into two small batches

Therefore, the makespan yielded by the case of transporting *u*+1 jobs with two batches is never greater than the makespan yielded by the case of transporting these jobs in one batch, because jobs transported in the first batch could be processed on machine 2 earlier than those shipped in one batch. Hence, the batch size in one shipment is always less than or equal to the threshold value *u*, and the capacity of the transporter is not necessarily greater than *u*. □

Assumption 3.2. The capacity of the transporter is greater than or equal to $\left\lceil \frac{2(t_1+t_2)}{p_1} \right\rceil - 1$.

Based on Assumption 3.1 and 3.2, a forward dynamic programming algorithm is proposed to solve the problem $TF_2 \mid p_{j1} \equiv p_1, v=1, c \geq u \mid C_{\max}$. According to Property 3.2, the size of the batch (denoted as *B*) which is greater than the threshold value *u* can always be divided into two small batches $\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$ and $B - \left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$, and yields a smaller makespan. If the number of jobs in the second batch ($B - \left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$) is still greater than *u*, this batch can be further split into two small batches $\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$ and $B - 2\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$ until the size of all these small batches is not greater than *u*. Inspired by this idea, a dynamic programming (DP) algorithm can be formulated as follows.

DP Algorithm for $TF_2 \mid p_{j1} \equiv p_1, v=1, c \geq u \mid C_{\max}$

- Optimal value function (OVF): *F(k)* = minimum completion time of a partial schedule containing the first *k* jobs, given that the completion time of job *k* is an integer departure point. (3.1)

- Arguments (ARG): k = index of a job such that the completion time of the job is an integer departure point. (3.2)
- Optimal policy function (OPF): j = number of jobs from integer departure point k to the previous integer departure point.
- Recurrence relation (RR):

$$F(k)=\min\left\{\begin{array}{ll} kp_1+t_1+\sum_{i=1}^{k} p_{i2}, & \text{if } k\le\lceil (t_1+t_2)/p_1\rceil \\ \min_{\lceil (t_1+t_2)/p_1\rceil\le j\le k-1}\left\{F(k-j)+C(k-j+1,k)\right\}, & \text{o.w.}\end{array}\right\}, k=1,2,...,n \tag{3.3}$$

where $C(k-j+1,\ k)$ is the minimum increase of the makespan due to jobs $k-j+1$ to k. It can be calculated by the following procedures:

Step 1. Let $g=1,\ x=k-j+1,\ x_0=k-j,\ t_0=p_1(k-j)$ and $C_0=F(k-j)$.

Step 2. $C_g=\max\{C_{g-1},t_0+g(t_1+t_2)+t_1\}+\sum_{i=x}^{x_0+\left\lfloor \frac{g(t_1+t_2)}{p_1}\right\rfloor} p_{i2}$.

Step 3. $x=x+\left\lfloor \dfrac{g(t_1+t_2)}{p_1}\right\rfloor,\ g=g+1$.

Step 4. If $g=\left\lfloor \dfrac{jp_1}{t_1+t_2}\right\rfloor$ then stop, and go to Step 5. Otherwise, go to Step 2.

Step 5. $C(k-j+1,k)=\max\{C_{g-1},kp_1+t_1\}+\sum_{i=x}^{k} p_{i2}-C_0$.

- Boundary Condition (BC): the *BC* has been included in *RR*.
- Answer (ANS):

$$\hat{F}(n)=\min\left\{\begin{array}{c}\min_{\lceil t_1+t_2/p_1\rceil\le j\le n-1}\left\{F(n-j)+\hat{C}(n-j+1,n)\right\} \\ F(n)\end{array}\right\} \tag{3.4}$$

where the calculation of $\hat{C}(x,y)$ is similar to that of $C(x, y)$, but it requires modifications on Step 2, 3, 4 and 5 as follows:

Step 2. $C_g = \max\{C_{g-1}, t_0 + g(t_1+t_2) + t_1\} + \sum_{i=x}^{\min\{n, x_0 + \left\lfloor \frac{g(t_1+t_2)}{p_1} \right\rfloor\}} p_{i2}$.

Step 3. $x = \min\{n, x + \left\lfloor \frac{g(t_1+t_2)}{p_1} \right\rfloor\}$, $g = g+1$.

Step 4. If $(g-1)(t_1+t_2) \geq jp_1$ then stop, and go to Step 5. Otherwise, go to Step 2.

Step 5. $\hat{C}(n-j+1, n) = C_{g-1} - C_0$.

The method to calculate C(x, y) is a deterministic procedure, because the departure points between stages (integer departure points) are all immediate departure points. It can be stated as the following property.

Property 3.3. Given two consecutive integer departure points d_j and d_k, and let d_j and d_k be the completion times of job j and job k on machine 1, respectively. Between these two integer departure points, once the transporter returns back to machine 1, it will transport the completed jobs immediately to machine 2 until its returning time to machine 1 is greater than $d_k - (t_1+t_2)$.

Proof: The number of jobs between these two integer departure points is $k-j$. According to Property 3.2, when the number of jobs in one batch is greater than u, this batch can be split into two small batches. Assume $k-j$ is greater than u. Then, these jobs can be divided into two batches $\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$ and $(k-j) - \left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$. Because the returning time of the transporter to machine 1 is $d_j + (t_1+t_2)$ which is greater than the completion time of the first batch $d_j + p_1\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$, the transporter will transport the first batch immediately to machine 2 so that this partition yields a smaller makespan. Similarly, the second batch can further divided into two batches if $(k-j)-\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$ is greater than *u*. This procedure is repeated until the last batch is less than or equal to *u*. Thus, between these two integer departure points, the transporter transport jobs from machine 1 to machine 2 immediately. □

Consequently, given two stages (integer departure points) *j* and *k*, the increase makespan $C(k-j+1,\ k)$ is determined. Also note that the formulation for the answer in Equation (3.4) is $\hat{F}(n)$ not $F(n)$ because it cannot be guaranteed that the completion time of the last job is an integer departure point. Therefore, this situation requires extra calculation in $\hat{F}(n)$ when the returning of the transporter to machine 1 is after the completion time of the last job on the first machine. Since the number of stages in this proposed algorithm is much less than that of the Lee and Chen's algorithm, this procedure is expected to be more efficient. A computational analysis and comparison on the computational efforts of the improved algorithms with Lee and Chen's (2001) are conducted in next section.

3.3 Complexity Analysis

According to Lee and Chen (2001), the time complexity of their algorithm is $O((cn)^3)$, where *c* is the capacity of the transporter and *n* is the number of jobs. To obtain the complexity of the proposed algorithm, the worst case ($k=1,\ldots,n$) is considered such that there are a total of *n* possibilities of *k*. For a given *k*, there are *k* possibilities of *j* since $j=1,\ldots,k$. Given *k* and *j*, there are at most *j* immediate departure points. In addition, in Boundary Condition (*BC*) we have

$j=1,\dots,n$ and there are at most *j* immediate departure points for a given *j*. Hence the overall complexity can be calculated as follows:

$$\begin{aligned}\sum_{k=1}^{n}\sum_{j=1}^{k} j &= \sum_{k=1}^{n}\frac{k(k+1)}{2}\\ &= \sum_{k=1}^{n}\left(\frac{k^2}{2}+\frac{k}{2}\right)\\ &= \frac{1}{2}\frac{n(n+1)(2n+1)}{6}+\frac{1}{2}\frac{n(n+1)}{2}\\ &= \frac{n(n+1)(n+2)}{6}.\end{aligned}$$

Thus, the complexity of the proposed algorithm is $O(n^3)$ which is better than $O(c^3n^3)$.

To illustrate the algorithm by an example, assume $k=20$ and $\left\lceil \frac{t_1+t_2}{p_1} \right\rceil = 4$, the minimum makespan of these jobs from 1 to 20 can be calculated as $F(20)=\min_{4\le j\le 19}\{F(20-j)+C(20-j+1,20)\}$. The makespans of all states for *j* ($4\le j\le 19$) should be calculated and the one with minimum value will be chosen. As an example, for *j*=10, the makespan is represented as $F(20-10)+C(20-10+1,20)=F(10)+C(11,20)$ where $F(10)$ is the minimum makespan for job 1 to 10 and has been obtained in previous stage. $C(11,20)$ is the minimum increase on the makespan due to jobs 11 to 20.

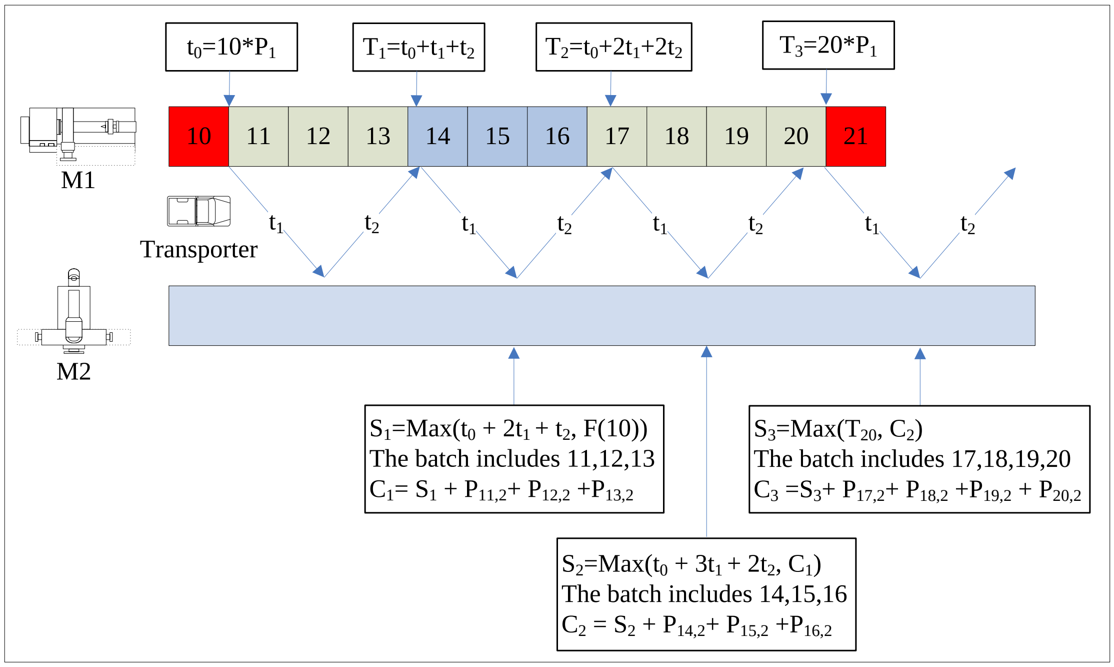

t₀=10*P₁
T₁=t₀+t₁+t₂
T₂=t₀+2t₁+2t₂
T₃=20*P₁
10
11
12
13
14
15
16
17
18
19
20
21
M1
Transporter
t₁
t₂
M2
S₁=Max(t₀ + 2t₁ + t₂, F(10))
The batch includes 11,12,13
C₁= S₁ + P₁₁,₂+ P₁₂,₂ +P₁₃,₂
S₂=Max(t₀ + 3t₁ + 2t₂, C₁)
The batch includes 14,15,16
C₂ = S₂ + P₁₄,₂+ P₁₅,₂ +P₁₆,₂
S₃=Max(T₂₀, C₂)
The batch includes 17,18,19,20
C₃ =S₃+ P₁₇,₂+ P₁₈,₂ +P₁₉,₂ + P₂₀,₂

Figure 3.3. An example with *k*=20 and *j*=10

Suppose that the transporter returns to machine 1 at T_1. At this time point, because $\left\lfloor \frac{t_1+t_2}{p_1} \right\rfloor$ is equal to 3, job 11, 12, and 13 are finished on machine 1 and ready for delivery to machine 2. The transporter delivers these jobs and departs from machine 1 immediately. The shipment will arrive at machine 2 at T_1+t_1. The jobs in this batch will be processed when machine 2 becomes available. Hence, the possible starting time (denoted by S_1) is either the arrival time of the batch or the completion time of the previous batch which is $F(10)$. As a result, the completion time of this batch is the starting processing time S_1 plus the total processing time of this batch on machine 2. Similarly, the second batch includes job 14, 15 and 16 which is delivered at T_2 ($T_2 = T_1 + t_1 + t_2$) and arrives machine 2 at $T_2 + t_1$. The completion time of the second batch is the S_2 plus the total processing time of the batch on machine 2.

Finally when the transporter returns to machine 1, job 20 is still under processing. Because the completion time of job 20 on machine 1 is the integer departure point, the transporter cannot depart until job 20 is finished. Hence, the last batch includes job 17, 18, 19, and 20 which arrives at machine 2 at $T_3 + t_1$. The completion time of the last batch can be obtained as above and the procedure to calculate $C(11, 20)$ is shown in Figure 3.3.

3.4 Concluding Summary

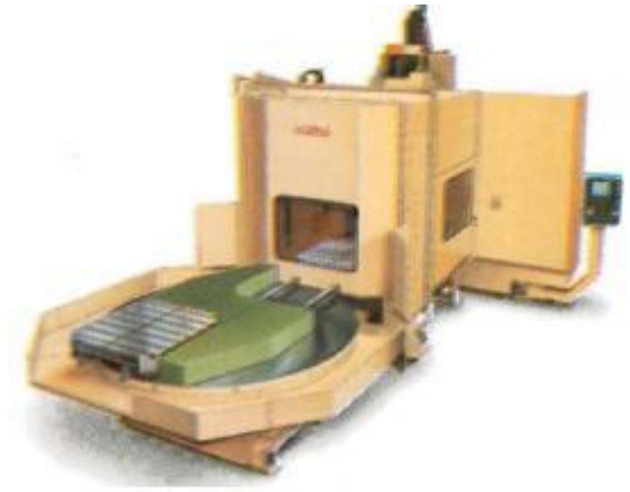
A special case for the *n*-job, two-machine, one transporter with a specific capacity flow shop problem with minimum makespan has been studied in this chapter. When processing times for all jobs on machine 1 are identical, a threshold value u of the transporter's capacity can be derived as Property 3.2. Under the assumptions of the identical processing time and the capacity of transporter is greater than or equal to *u*, the problem $TF_2 \mid p_{j1} \equiv p_1, v = 1, c \geq u \mid C_{\max}$ can be solved in polynomial time by the proposed dynamic programming algorithm. The computational complexity of the algorithm has been shown as $O(n^3)$ which is better than the algorithm proposed by Lee and Chen (2001). Therefore, when the capacity of the transporter is not less than u ($c \geq u$), the problem will be solved more efficiently by using the proposed algorithm.

Chapter 4

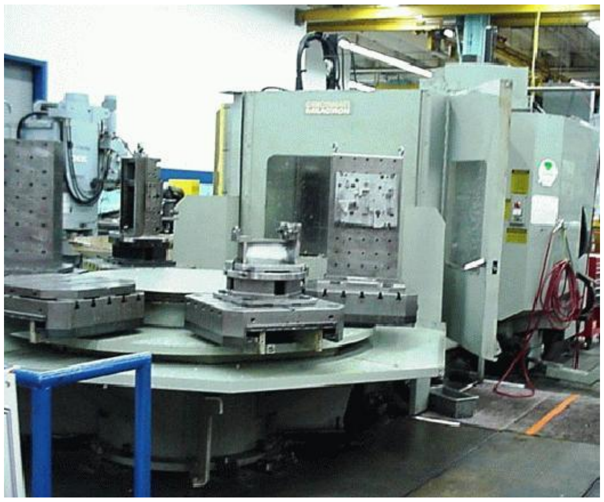
Flow Shop with Synchronous Material Movement

4.1 Introduction

Automated manufacturing systems which integrate material handling and processing devices are commonly employed in manufacturing industries to gain a competitive advantage. Typically, an integrated cell or machining center consists of an input/output station, one or more processing machines, and material handling devices to efficiently process and move a group of similar parts. The material handling devices can be robots, conveyors, automated guided vehicles, cranes and transporters. One type of material handling devices is called rotary table. The rotary table is surrounded by the loading/unloading station and processing machines. Jobs should be loaded on the table before being processed. Loaded jobs are transported to each processing machine sequentially by the rotary table. Once all these jobs complete their current operations, the rotary table rotates in a clockwise or counterclockwise direction to move these jobs simultaneously to their next processing machines. Examples of manufacturing cells integrated rotary tables are the T-line machining centers developed by Cincinnati Milacron (Milacron 1989). Figure 4.1 shows two models of these T-line machining centers.



(a) T-40 Machining Center



(b) T-20 Machining Center

Figure 4.1. Examples of Cincinnati Milacron T-line machining centers
(Sources: (a) Milacron 1989 (b) www.locatoronline.com/machinery/detail.cfm?adid=321905)

In a T-line machining center, there are one loading/unloading (L/U) station and a number of computer numerical control (CNC) machines which surrounds a rotary table. Fixtures are installed on the rotary table called pallets which are used to carry and stabilize jobs. Before being processed, a job is loaded onto one of pallets at the L/U station. It is processed sequentially through a number of CNC machines and finally unloaded from the machining center at the L/U station. Jobs on the rotary table are transported to the L/U station and machines simultaneously. Figure 4.2 below shows a T-line machining center with two CNC machines and a rotary table with three pallets.

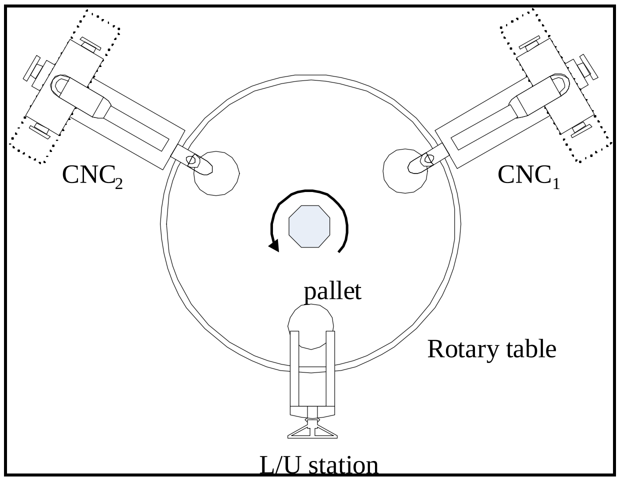


Figure 4.2. A T-line machining center with two CNC machines

In this setting, while two jobs, say A and B, loaded in two pallets are being processed by CNC_2 and CNC_1, respectively, another job, say C, is concurrently being loaded onto the third pallet at the L/U station. After the processing operations on both machines and loading operation are completed, the table rotates 120 degrees counterclockwise such that job C is transported to CNC_1, and job B to CNC_2. Finished job A will be unloaded from the pallet at the L/U station and a new job, say D, can be loaded on the pallet.

Transporting jobs simultaneously is referred to as synchronous material movement in this study. This study focuses on sequencing jobs in this particular type of machining centers. Efficiency is one of the desirable characteristics of these machining centers, and minimizing the makespan is equivalent to maximizing utilization of a machining center. Thus, the objective of this study is to find a job sequence that minimizes the makespan. The three-field notation introduction by Graham *et al.* (1979) is adopted to represent the problem. As mentioned in Chapter 2, *synmv* represents the characteristic of synchronous material movement and *re-LU* indicates that a job has to be loaded and unloaded at the same L/U station. Hence, the scheduling problem with makespan objective for a T-line machining center with *m* machines can be denoted as $F_m \mid synmv, re-LU \mid C_{\max}$. In this chapter, the scheduling problem with one CNC machine in a T-line machining center will be investigated.

This chapter is divided into six sections. In Section 2, the setting of the machining center with one machine is described and the scheduling problem of the machining center is shown to be NP-hard in the strong sense. Section 3 provides a polynomial algorithm for a special case of the problem with a constant loading time or unloading time. Section 4 includes a proposed dynamic programming algorithm for the problem, the analysis of its computational effort and a numerical example to demonstrate the algorithm. In Section 5, two heuristic algorithms are developed to construct near optimal solutions for large-scale versions of this problem. Section 6 examines the performance of the proposed algorithms on several settings.

4.2 T-line Machining Center with One CNC Machine

The T-line machining center consists of one CNC machine, a loading/unloading station, denoted as L/U, and a rotary table with two pallets. This setting is illustrated in Figure 4.3. First, a job has to be loaded onto a pallet before processing. After it is loaded, the job is transported to the CNC machine by the rotary table. The job will finally return back to the L/U station and be unloaded from the machining center. One pallet can only contain one job and the CNC machine can only process one job at a time. No preemption is allowed. Assume there are *n* jobs that have to be processed by the machining center. All jobs are available at time zero and wait in the input buffer of the L/U station. For the makespan criterion, the optimal sequence is independent of the rotation time of the rotary table; thus, the rotation time can be neglected. The loading, processing, and unloading times for job j are denoted as l_j, p_j, and u_j, respectively. The problem is to determine a job sequence that yields the minimum makespan. Thus, the notation for this scheduling problem is represented as $F_1 \mid synmv, re-LU \mid C_{\max}$.

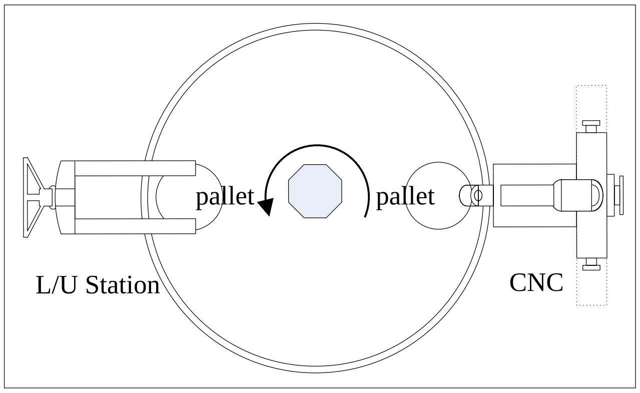


Figure 4.3. A T-line machining center with one CNC machine

Figure 4.4 illustrates a schedule of jobs at each station. A time period between two successive rotations of the rotary table is defined as a cycle time, called C_i where $i = 1, \ldots, n+2$. In the first two cycles, there are only loading operations performed at the L/U station. Similarly, only unloading operations are required in the last two cycles. From cycle 3 to cycle n, a job should be unloaded from the rotary table first before a new job can be loaded. Because the rotary table does not rotate until the completions of all operations are performed at each station, a cycle time is equal to the largest operation time among the corresponding operations currently performed at each station. Thus, each cycle time can be represented as follows given a job sequence $J_{[1]}$, $J_{[2]}$, …, $J_{[n]}$ where the notation $J_{[i]}$ represents the job is sequenced in position i:

- $C_1 = l_{[1]}$; $C_2 = \max\{p_{[1]},\ l_{[2]}\}$, (4.1)
- $C_i = \max\{p_{[i-1]},\ u_{[i-2]} + l_{[i]}\}$, $i = 3, \ldots, n$, (4.2)
- $C_{n+1} = \max\{p_{[n]},\ u_{[n-1]}\}$; $C_{n+2} = u_{[n]}$. (4.3)

where $l_{[i]}$, $p_{[i]}$ and $u_{[i]}$ are the loading, processing and unloading times of the job sequenced in position i, respectively.

Therefore, the makespan of the sequence is the summation of all cycle times formulated as $\sum_{i=1}^{n+2} C_i$. The objective of the problem is to find a sequence which minimizes this value.

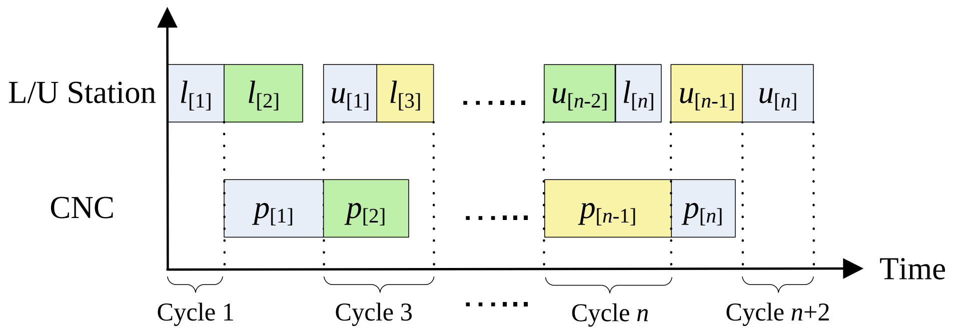


Figure 4.4. Schedule of jobs at each station in a one-CNC T-line machining center

Computational Complexity

Discovering the computational complexity of a problem is the first step in working on a problem. This knowledge can help direct the problem-solving process towards the development of more useful algorithms. As a result, the complexity of the problem with one CNC machine is explored first. Logendran and Sriskandarajah (1993) consider a two-machine flow shop scheduling problem with blocking and anticipatory setup. Setup on these two machines can be performed in anticipation of arriving jobs when the machines are idle. They show the problem is strongly NP-hard by reducing it to an existing NP-hard problem – “Numerical Matching Problem with Target Sums.” Inspired by their research, the problem $F_1 | synmv, re-LU | C_{\max}$ is also proven to be NP-hard in the strong sense by showing the equivalence to the numerical matching problem with target sums.

Numerical matching problem with target sums (Garey and Johnson 1979)

Given two disjoint sets of positive integers $A=\{a_1,a_2,\ldots,a_n\}$ and $B=\{b_1,b_2,\ldots,b_n\}$, and a target set with positive integers $C=\{c_1,c_2,\ldots,c_n\}$, the numerical matching problem with target sums is to answer the following question:

Question: Can $A \bigcup B$ be partitioned into n disjoint sets D_k, each containing exactly one element from set A and one element from set B, such that, $c_k = a_{[k]} + b_{[k]},\ a_{[k]}, b_{[k]} \in D_k$ for $k = 1,\ldots,n$?

Below, the problem $F_1 \mid synmv, re-LU \mid C_{\max}$ is shown to be NP-hard by reducing it to the numerical matching problem with target sums in polynomial time and also by showing the equivalence of these two problems in opposite direction. First, the problem $F_1 \mid synmv, re-LU \mid C_{\max}$ is restated as a decision problem as follows:

Decision problem (*P*). Given a number of jobs with loading, processing and unloading times that have to be processed in a one-CNC T-line machining center, does there exist a schedule, say σ, with makespan $C_{max}(\sigma)$ which is less than or equal to a given value Z ?

Given an arbitrary instance of the numerical matching problem with target sums defined by positive integers $a_i,\ b_i$ and $c_i,\ i=1,\ldots,n$, construct an instance (denoted as Instance P1) of Problem P with 3n jobs as follows:

$J = \{ \mathrm{Ja}_i,\ \mathrm{Jb}_i,\ \mathrm{Jc}_i \mid i = 1,\dots,n\}$, J is a set containing three types of jobs Ja, Jb and Jc and the number of jobs for each type is n. The loading time (l), processing time (p), and unloading time (u) of each job in each type are defined as:

Ja-type: $l(\mathrm{Ja}_i) = 1,\ p(\mathrm{Ja}_i) = 3,\ u(\mathrm{Ja}_i) = 2\beta + 3a_i,\ 1 \le i \le n,$

Jb-type: $l(\mathrm{Jb}_i) = \beta + 3b_i,\ p(\mathrm{Jb}_i) = \beta + 1,\ u(\mathrm{Jb}_i) = 1,\ 1 \le i \le n,$

Jc-type: $l(\mathrm{Jc}_i) = 2,\ p(\mathrm{Jc}_i) = 3\beta + 3c_i,\ u(\mathrm{Jc}_i) = \beta,\ 1 \le i \le n,$ where $\beta = 3\max\{c_i \mid 1 \le i \le n\}$.

In addition, the target value Z is defined as:

$$Z = 2 + 4n + 4n\beta + 3C_0 \text{ where } C_0 = \sum_{i=1}^{n} c_i .$$

Lemma 4.1. If there exists a solution for the numerical matching problem with target sums, then the makespan of schedule σ for Instance P1 constructed as shown in Figure 4.5, is equal to the target value Z.

Proof. By assumption, a solution to the numerical matching problem with target sums exists so that $c_{[k]} = a_{[k]} + b_{[k]}$ where $a_{[k]}, b_{[k]}$ and $c_{[k]}$ are the elements in D_k , $k = 1,\dots,n$. Since each $a_{[k]}, b_{[k]}$ and $c_{[k]}$ correspond to a job of type Ja, type Jb, and type Jc, respectively, a set (denoted as B_k) containing three jobs can be formed based on each D_k. Consider the following schedule σ (see Figure 4.5):

$$\sigma = [B_1, B_2, \dots, B_n] \text{ where jobs in } B_k \text{ are sequenced as } [\mathrm{Ja}_{[k]},\ \mathrm{Jc}_{[k]},\ \mathrm{Jb}_{[k]}] .$$

The idle time of the CNC machine only occurs at the first and last cycle and the summation of the idle time is 2 units of time. Hence, the makespan of schedule σ can be calculated by adding idle times and busy times of the CNC machine:

$$C_{\max} = 2 + 3n + 3n\beta + 3\sum_{i=1}^{n} c_i + n\beta + n = 2 + 4n + 4n\beta + 3C_0 = Z\,. \qquad \square$$

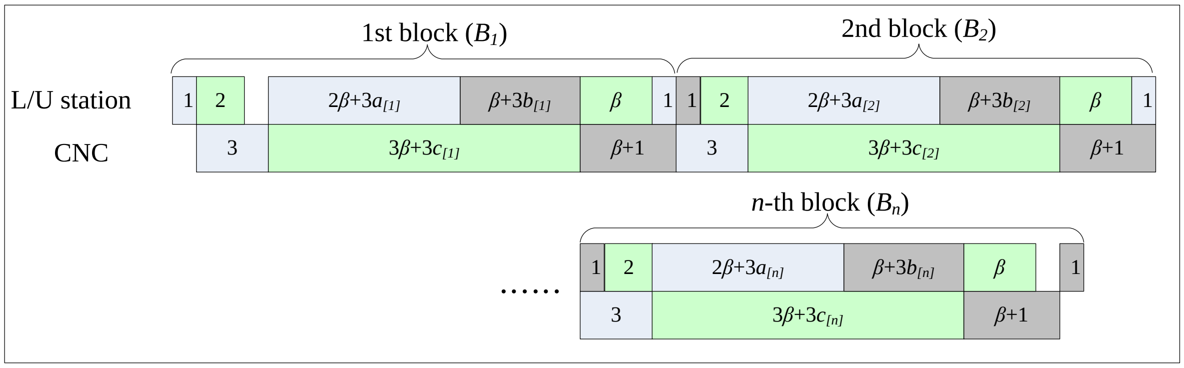


Figure 4.5. Schedule σ on the L/U station and CNC machine

Lemma 4.2. If there is an optimal sequence σ for Instance P1 with the makespan $C_{\max}(\sigma) \le Z$, then the CNC machine is busy on processing jobs except one unit of idle time at the first and last cycles.

Proof. The total processing time on the CNC machine is $Z-2$ which can be calculated as:

$$\sum_{i=1}^{n} p(Ja_i) + p(Jb_i) + p(Jc_i) = 3n + n(\beta+1) + \sum_{i=1}^{n} 3\beta + 3c_i$$

$$= 4n + 4n\beta + \sum_{i=1}^{n} 3c_i = 4n + 4n\beta + 3C_0 = Z - 2$$

Because of the makespan $C_{\max}(\sigma) \le Z$, the idle time on the CNC machine is 2 units of time at most. In addition, there is no operation on the CNC machine when the first job is loaded and the last job is unloaded on the L/U station. Since the smallest loading and unloading times are equal to 1 respectively, the idle time can only occur at the first and last cycles on the CNC machine. $\square$

Lemma 4.3. If there is an optimal sequence σ for Instance P1 with the makespan $C_{\max}(\sigma) \le Z$, then the L/U station is busy processing jobs except one unit of idle time at the second and second last cycles.

Proof. The total operation time on the L/U station is $Z-2$ which can be calculated as:

$$\sum_{i=1}^{n}[l(Ja_i)+l(Jb_i)+l(Jc_i)+u(Ja_i)+u(Jb_i)+u(Jc_i)]$$

$$=n+\sum_{i=1}^{n}(\beta+3b_i)+2n+\sum_{i=1}^{n}(2\beta+3a_i)+n+n\beta$$

$$=4n+4n\beta+3\sum_{i=1}^{n}(a_i+b_i)=4n+4n\beta+3C_0=Z-2$$

Because of the makespan $C_{\max}(\sigma) \le Z$, the idle time on the L/U station is 2 units of time at most. In the second cycle, there is no job required to be unloaded. Therefore, there is only one job being loaded at the L/U station, and one job being processed on the CNC machine. Since the CNC machine cannot be idle except for in the first and last cycles, the processing time should be greater than or equal to the loading time in the second cycle. By considering all combinations of processing and loading times, the minimum idle time at the L/U station can only be 1 in the second cycle. Likewise, no job will be loaded in the second last cycle. There is only one job being unloaded on the L/U station and one job being processed on the CNC machine. The minimum idle time at the L/U station is also equal to 1. Thus, one unit of idle time can only occur at the second and second last cycles at the L/U station. □

Lemma 4.4. If there is an optimal sequence σ for Instance P1 with the makespan $C_{\max}(\sigma) \le Z$ and the jobs in sequence σ are partitioned into n disjoint blocks B_k where $B_k=\{J_{[3k-2]}, J_{[3k-1]}, J_{[3k]}\}$, $k=1,\ldots,n$, then jobs $J_{[3k-2]}$, $J_{[3k-1]}$, and $J_{[3k]}$ in B_k are Ja-type, Jc-type and Jb-type, respectively.

Proof. First, we show that the first three jobs $J_{[1]}, J_{[2]}$, and $J_{[3]}$ in B_1 are Ja-type, Jc-type and Jb-type. Three cases should be considered:

- Case 1: $J_{[1]}$ is not Ja-type.

  If $J_{[1]}$ is Jb-type, the idle time on the CNC machine is greater than 1, which contradicts Lemma 4.2 as shown in Figure 4.6.

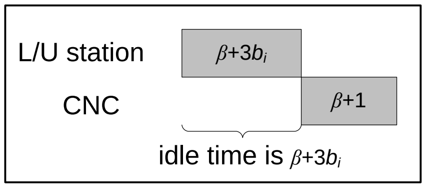


Figure 4.6. A Jb-type job is sequenced in the first position

  If $J_{[1]}$ is Jc-type, the idle time on the CNC machine is also greater than 1, which contradicts Lemma 4.2 as shown in Figure 4.7.

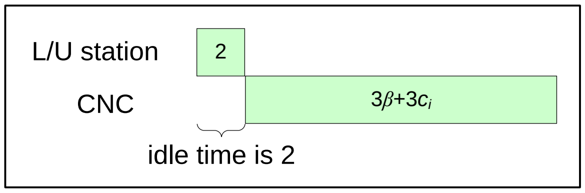


Figure 4.7. A Jc-type job is sequenced in the first position

  Thus, the job sequenced in the first position has to be Ja-type.

- Case 2: $J_{[2]}$ is Jc-type.

  If $J_{[2]}$ is Ja-type, the idle time on the L/U station is 2, which contradicts Lemma 4.3 as shown in Figure 4.8.

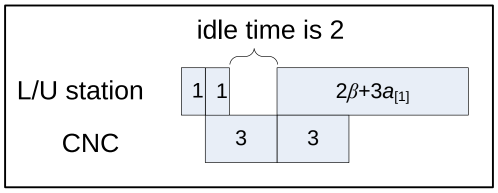


Figure 4.8. A Ja-type job is sequenced in the second position

If $J_{[2]}$ is Jb-type, the idle time on the CNC machine in the second cycle is greater than 0, which contradicts Lemma 4.2 as shown in Figure 4.9.

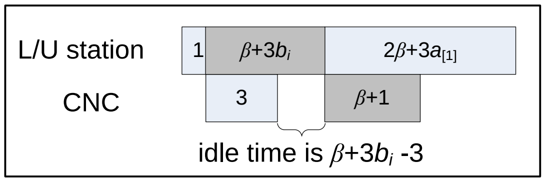


Figure 4.9. A Jb-type job is sequenced in the second position

Thus, the job sequenced in the second position has to be Jc-type.

- Case 3: $J_{[3]}$ is Jb-type.

If $J_{[3]}$ is Ja-type, the idle time on the L/U station in the third cycle is greater than 0, which contradicts Lemma 4.3 as shown in Figure 4.10.

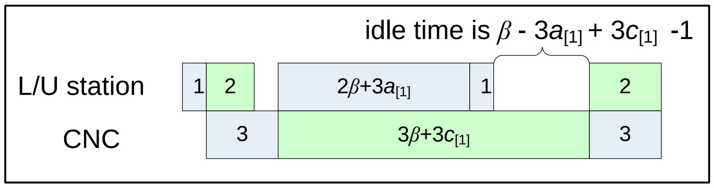


Figure 4.10. A Ja-type job is sequenced in the third position

If $J_{[3]}$ is Jc-type, the idle time on the L/U station in the third cycle is greater than 0, which contradicts Lemma 4.3 as shown in Figure 4.11.

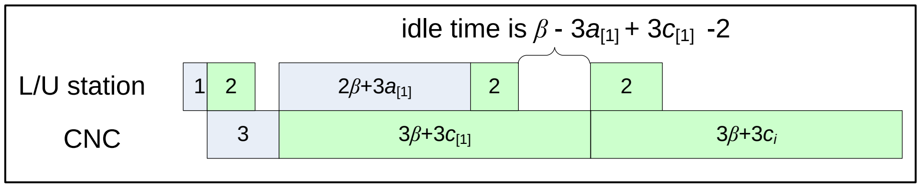


Figure 4.11. A Jc-type job is sequenced in the third position

Thus, the job sequenced in the third position has to be Jb-type. Analogously, for $k = 2,\ldots,n$, we can also show that jobs $J_{[3k-2]}, J_{[3k-1]}$, and $J_{[3k]}$ in B_k are Ja-type, Jc-type and Jb-type, respectively. □

Since jobs $J_{[3k-2]}, J_{[3k-1]}$, and $J_{[3k]}$ in each B_k correspond to jobs of type Ja, Jc and Jb, schedule σ can be represented as in the following form:

$$[\text{Ja}_{[1]},\ \text{Jc}_{[1]},\ \text{Jb}_{[1]}, \text{Ja}_{[2]},\ \text{Jc}_{[2]},\ \text{Jb}_{[2]},\ldots,\text{Ja}_{[n]},\ \text{Jc}_{[n]},\ \text{Jb}_{[n]}]$$

Lemma 4.5. If there is an optimal sequence σ for Instance P1 with the makespan $C_{\max}(\sigma) \le Z$ and jobs in optimal schedule σ are partitioned into n blocks B_k where jobs in B_k are sequenced as $[\text{Ja}_{[k]},\ \text{Jc}_{[k]},\ \text{Jb}_{[k]}]$, then $a_{[k]} + b_{[k]} = c_{[k]},\ k = 1,\ldots,n$.

Proof. First, we show that for the first three jobs in B_1, the processing time of $\text{Jc}_{[1]}$ is equal to the sum of the unloading time of $\text{Ja}_{[1]}$ and the loading time of $\text{Jb}_{[1]}$. That is $a_{[1]} + b_{[1]} = c_{[1]}$. Two cases should be considered:

- Case 1: assume $a_{[1]} + b_{[1]} < c_{[1]}$. Then, the idle time on the L/U station in the third cycle is greater than 0 as shown in Figure 4.12, which contradicts Lemma 4.3.

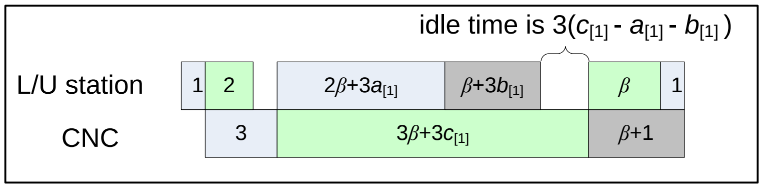


Figure 4.12. Idle time on the L/U station in the third cycle when $a_{[1]} + b_{[1]} < c_{[1]}$

- Case 2: assume $a_{[1]} + b_{[1]} > c_{[1]}$. Then, the idle time on the CNC machine in the third cycle is greater than 0 as shown in Figure 4.13, which contradicts Lemma 4.2, and the claim is justified.

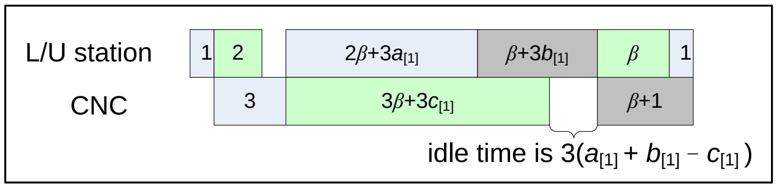


Figure 4.13. Idle time on the CNC machine in the third cycle when $a_{[1]} + b_{[1]} > c_{[1]}$

Similarly, the equality $a_{[k]} + b_{[k]} = c_{[k]}$ can be concluded for B_k , $k = 2, \ldots, n$. $\square$

Theorem 4.1. The scheduling problem with the makespan objective in a T-line machining center with one CNC machine is NP-hard in the strong sense.

Proof. To complete the proof, we have to show that there exists a schedule σ for Problem P **if and only if** there is a solution to the numerical matching problem with target sums. For the **if**

part, the numerical matching problem with target sums is equivalent to Problem *P* based on lemma 4.1. For the **only if** part, Problem *P* can be also converted to the numerical matching problem with target sums from Lemmas 4.2 to 4.5. Furthermore, it only takes polynomial time to construct Instance P1 from an instance of the numerical matching problem with target sums. □

4.3 Special Case – Constant Unloading Time

Typically, the operation of removing or unloading a job from a machining center is simple. Therefore, it is reasonable to consider a scenario with a common unloading time (denoted as *c*) in a T-line machining center. In this case, the operation time at the L/U station can be regarded as a longer loading time without unloading time from cycle 3 to cycle *n* given *n* is the number of jobs as shown in Figure 4.14. Without considering the first two loading operations and the last two unloading operations for a given sequence, this problem becomes a two-machine flow shop problem with blocking, which is polynomial solvable by the Gilmore-Gomory algorithm (Gilmore and Gomory 1964). In this section, the scheduling problem of a one-machine T-line machining center with constant unloading time (denoted as $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$) will be shown to be polynomial solvable, and an algorithm to solve the problem will provided.

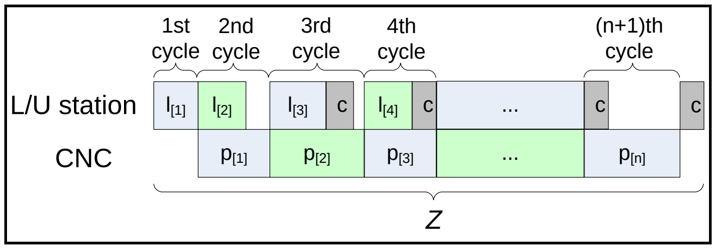


Figure 4.14. Schedule in a one-machine T-line machining center with constant unloading time

Assume an optimal sequence for the problem $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$ is J$_{[1]}$, J$_{[2]}$, …, J$_{[n]}$, and the optimal makespan is Z. From Figure 4.14, we observe that the operation time at the L/U station for each cycle is the sum of the loading time and constant c from the 3rd cycle to nth cycle. Therefore, we consider J$_{[3]}$ to J$_{[n]}$ and add constant c to the loading times of these jobs as the operation times on the L/U station. Alternatively, $l_{[j]} + c$ can be regarded as a new loading times for $J_{[j]}, 3 \le j \le n$, and no unloading operations is required. Then, this problem becomes a two-machine flow shop problem with blocking as shown in Figure 4.15, and the problem can be solved by the Gilmore-Gomory algorithm. Since the Gilmore-Gomory algorithm will be frequently applied in this research, the procedures of this algorithm and a numerical example are presented in the Appendix.

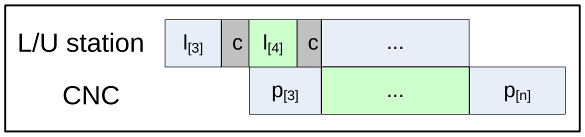


Figure 4.15. Two-machine flow shop problem with blocking for $J_{[j]}, 3 \le j \le n$

However, in order to maintain the optimal sequence in the two-machine flow shop problem with blocking problem as the same as the optimal sequence in the original problem, two jobs should be added. One job is denoted as J$_0$ and the other denoted as J$_{n+1}$. The loading times of J$_0$ and J$_{n+1}$ are set to be 0 and c, respectively. The processing time of J$_0$ and J$_{n+1}$ are set to be $p_{[2]}$ and 0, respectively. In addition, J$_0$ must be sequenced in the first position of the optimal sequence and J$_{n+1}$ must be sequenced in the last position. Assume the optimal makespan for the modified problem is Z^G as shown in Figure 4.16. Then, the optimal makespan for the problem

$F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$ has to be shown that it is equal to $Z^G + l_{[1]} + \max(l_{[2]},\ p_{[1]}) + c$. Namely, given the first two jobs in an optimal sequence, the remaining sequence can be generated by applying the Gilmore-Gomory algorithm to the modified problem.

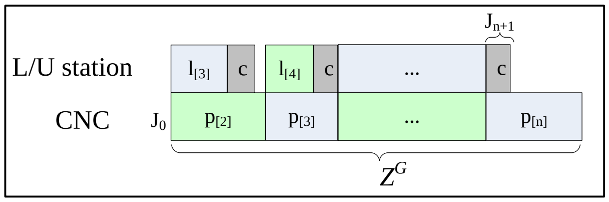


Figure 4.16. The makespan *Z'* for the modified problem

Two-machine flow shop problem with blocking ($F_2 \mid \text{blocking} \mid C_{\max}$ **)**

Define the modified problem (denoted as Problem *G*) as follows:

Given the first two jobs in an optimal sequence of the scheduling problem in a one-CNC T-line machining center, let the loading time of J$_{[j]}$ be $l_{[j]} + c$; $j = 3, \ldots, n$. Add two additional jobs J$_0$ and J$_{n+1}$ and their loading times and processing times are defined as $l_0 = 0,\ p_0 = p_{[2]},\ l_{n+1} = c$ and $p_{n+1} = 0$. Problem *G* is to find an optimal sequence of these jobs ($\mathrm{J}_{[j]}, j = 3, \ldots, n$) and J$_0$ and J$_{n+1}$. The optimal makespan obtained by applying the Gilmore-Gomory algorithm to Problem *G* is denoted as Z^G.

Lemma 4.6. J$_0$ must be in the first position of the optimal sequence for Problem *G*.

Proof: Assume J$_0$ is not in the first position of the optimal sequence, and the optimal makespan is Z^1. The jobs sequenced before J$_0$ form Block 1 and the remaining jobs including J$_0$ form Block 2, as shown in Figure 4.17. In addition, the loading time of the job in the first position is denoted as l_a and the processing time of the job right before J$_0$ and of the job in the last position are denoted as p_b and p_c, respectively.

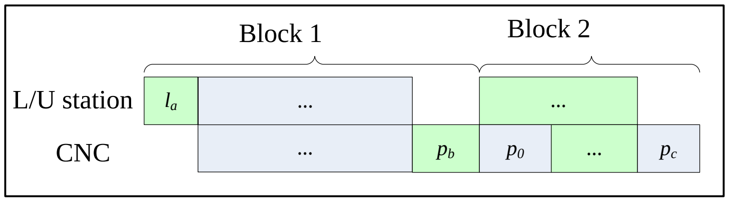


Figure 4.17. J$_0$ is not sequenced in the first position for Problem *G*

The sequence of jobs in Block 1 is exchanged with that in Block 2 as shown in Figure 4.18. Let the makespan of the new sequence be Z^2.

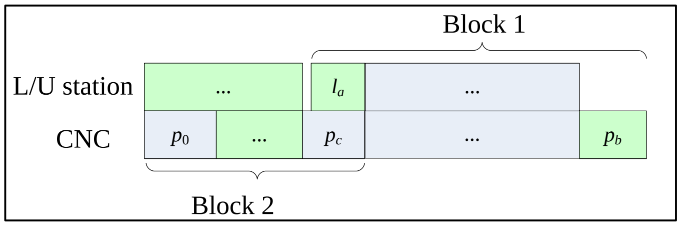


Figure 4.18. J$_0$ is sequenced in the first position for Problem *G*

The difference of the makespans between these two sequences is:

$$Z^1 - Z^2 = l_a + p_b + p_c - \max(l_a,\ p_c) - p_b = l_a + p_c - \max(l_a,\ p_c) > 0\,.$$

Z^2 is strictly less than Z^1 which contradicts the assumption that Z^1 is the optimal makespan. Hence, J$_0$ must be sequenced in the first position of the optimal sequence. □

Lemma 4.7. J$_{n+1}$ must be in the last position of the optimal sequence for Problem *G*.

Proof: Assume J$_{n+1}$ is not in the last position of the optimal sequence, and the optimal makespan is Z^1. The jobs sequenced after J$_{n+1}$ form Block 2, and the remaining jobs including J$_{n+1}$ form

Block 1, as shown in Figure 4.19. In addition, the loading time of the job in the first position and of the job right behind J$_{n+1}$ are denoted as l_a and l_c, respectively. The processing time of the job right before J$_{n+1}$ and of the job in the last position are denoted as p_b and p_d, respectively.

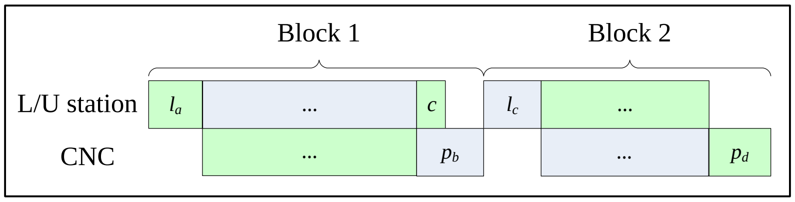


Figure 4.19. J$_{n+1}$ is not sequenced in the last position for Problem *G*

The sequence of jobs in Block 1 is exchanged with that in Block 2 as shown in Figure 4.20. Let the makespan of the new sequence be Z^2.

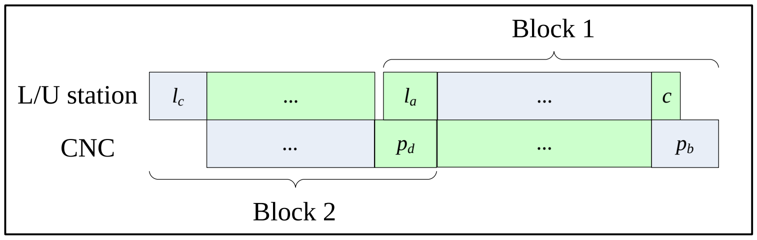


Figure 4.20. J$_{n+1}$ is sequenced in the last position for Problem *G*

The difference of the makespans between these two sequences is:

$$Z^1 - Z^2 = l_a + \max(c,\ p_b) + l_c + p_d - l_c - \max(l_a,\ p_d) - \max(c,\ p_b)$$

$$= l_a + p_d - \max(l_a,\ p_d) > 0.$$

Z^2 is strictly less than Z^1 which contradicts the assumption of Z^1 is the optimal makespan. Hence, J$_{n+1}$ must be sequenced in the last position of the optimal sequence. □

Theorem 4.2. The optimal makespan of the problem $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$ is $Z^G + l_{[1]} + \max(l_{[2]}, p_{[1]}) + c$ given the first two jobs in the optimal sequence where Z^G is the optimal makespan of Problem *G*.

Proof: Based on Lemma 4.6 and Lemma 4.7, J_0 must be sequenced in the first position and J_{n+1} must be sequenced in the last position of the optimal sequence. Suppose the optimal makespan (*Z*) for the original problem $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$ is less than $Z^G + l_{[1]} + \max(l_{[2]}, p_{[1]}) + c$. Remove the time period $l_{[1]} + \max(l_{[2]}, p_{[1]})$ at the beginning of the optimal sequence and time period *c* at the end from *Z* as illustrated in Figure 4.21. Then, the value $Z - l_{[1]} - \max(l_{[2]}, p_{[1]}) - c$ is less than Z^G which contradicts that Z^G is the optimal makespan obtained by the Gilmore-Gomory algorithm. Conversely, if *Z* is greater than $Z^G + l_{[1]} + \max(l_{[2]}, p_{[1]}) + c$, the value of *Z* can obviously be improved by replacing the sequence with the one obtained by the Gilmore-Gomory algorithm. Hence, the optimal makespan, *Z*, should be equal to $Z^G + l_{[1]} + \max(l_{[2]}, p_{[1]}) + c$, and Theorem 4.2 is proven. □

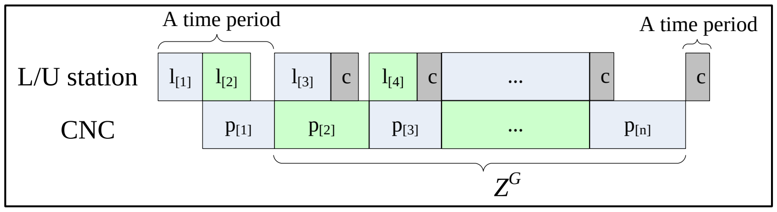


Figure 4.21. The makespan for the problem $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$

Based on Theorem 4.2, an algorithm can be developed to solve the problem with a common unloading time. Because no unloading operations are required at the first two cycles, the problem can't be solved just by applying the Gilmore-Gomory algorithm. This is the reason that the first two jobs in the optimal sequence are required as Theorem 4.2. Since the first two jobs in the optimal sequence are unknown, all possible combinations of the first two jobs have to be considered. For each combination of the first two jobs in the sequence, the Gilmore-Gomory algorithm is applied to obtain the optimal sequence for the rest of jobs, and the best makespan is recorded. Until all combinations are performed, the best makespan recorded will be the optimal makespan for the problem.

Algorithm for $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$

Step 1. Initialize the best makespan as infinite.

Step 2. For $j = 1$ to n and let J$_j$ be in the first position of the sequence.

Step 2.1. For $k = 1$ to n and $k \neq j$, and let J$_k$ be in the second position of the sequence.

Step 2.1.1. Let $p_{i1} = l_i + c$ and $p_{i2} = p_i$; $i = 1, \ldots, n$; $i \neq j$; $i \neq k$.

Step 2.1.2. Let the processing times of job J$_k$ as $p_{k1} = 0$ and $p_{k2} = p_k$. Add job J$_{n+1}$ with the processing times as $p_{n+1,1} = c$ and $p_{n+1,2} = 0$.

Step 2.1.3. Solve the problem as a two-machine flow shop problem with blocking by the Gilmore-Gomory algorithm, and obtain the optimal sequence and the makespan Z^G.

Step 2.1.4. The makespan for the original problem is $Z^G + l_j + \max(l_k,\ p_j) + c$.

Step 2.1.5. If the current makespan is less than the best makespan, then set the best makespan as the current makespan, and append J$_j$ at the beginning of the current sequence as the best sequence.

Step 3. Output the best makespan and the best sequence.

Since the number of combinations of the first and second jobs in a sequence are $n(n-1)$ given n is the number of jobs and the complexity of the Gilmore-Gomory algorithm is $O(n\log n)$, the complexity of the proposed algorithm is $O(n^3 \log n)$. The algorithm also can be applied to the problem with constant loading time.

4.4 Dynamic Programming Algorithm for $F_1 \mid synmv, re-LU \mid C_{\max}$

Dynamic programming is an efficient approach to obtain an optimal solution when the problem requires a sequence of interrelated decisions (Dreyfus and Law 1997). Obtaining the minimum makespan for the scheduling problem of a T-line machining center has the characteristic of requiring a sequence of interrelated decisions as Equations (4.1) to (4.3). Thus, a forward dynamic programming procedure is formulated to solve this problem. Given n jobs which have to be processed by the machining center and these jobs are numbered from 1 to n. Let $N = \{1, 2, \ldots, n\}$ be a set of jobs and let S be a subset of N containing the jobs that have already been processed in the machining center. Let g represent the job concurrently being processed on the CNC, and j be the job being loaded at the L/U station. Then the dynamic programming formulation is as follows.

DP Algorithm for $F_1 \mid synmv, re-LU \mid C_{\max}$

- Optimal value function (OVF): $f_i(S, g, j)$ = minimum completion time for processing job g on CNC, unloading the last job in S and loading job j at the L/U station, given that the i jobs in S have already been completed. (4.4)

- Optimal policy function (OPF): $p_i(S, g, j)$ = last job unloaded at the L/U station. Equivalently, this is also the last job added to set S. (4.5)

- Recurrence relation (RR):

$$f_i(S,g,j) = \min_{k \in S}\{f_{i-1}(S \setminus \{k\}, k, g) + \max\{p_g, u_k + l_j\}\};\ i = 1,\ldots,n-2 \quad ; \quad \{g, j\} \subseteq N \quad ;$$

$$S \subseteq N \setminus \{g, j\},\ |S| = i. \qquad (4.6)$$

- Boundary condition (BC): $f_0(\varnothing, g, j) = l_g + \max\{p_g, l_j\};\ \{g, j\} \subseteq N$. (4.7)

- Answer (ANS):

$$\min_{g \subseteq N}\{f_{n-1}(S, g, \varnothing)\} \qquad (4.8)$$

where $f_{n-1}(S, g, \varnothing) = \min_{k \in S}\{f_{n-2}(S \setminus \{k\}, k, g) + \max\{p_g, u_k\}\} + u_g \quad ; \quad g \subseteq N \quad ; \quad S = N \setminus \{g\}$,

$|S| = n+1$. (4.9)

Computational Effort Analysis

The computational effort of the dynamic programming algorithm is evaluated by the number of operations performed: "Addition" and "Comparison." The number of operations required for each stage of the algorithm is summarized in Table 4.1.

Table 4.1: Number of operations required for each stage

Stage		Number of combinations	Addition	Comparison
Boundary condition ($i = 0$)		$n(n-1)$	1	1
Recurrence relation ($1 \le i \le n-2$)		$n(n-1)C_i^{n-2}$	$2i$	$i + (i - 1)$
Answer	f_{n-1} ($i = n - 1$)	n	n	$n - 1 + n - 2$
	Minimum makespan	1	0	$n - 1$

In boundary condition, there are $n(n-1)$ combinations for job g and j. Each combination requires one addition and one comparison to obtain the value for f_0. In recurrence relation, for each i, there are $n(n-1)$ choices for jobs g and j, and C_i^{n-2} combinations of jobs in set S. Each combination of (S, g, j) has i candidates in set S for k. To obtain the minimum value among these i candidates, it requires extra $i-1$ comparisons. In the answer formulation, there are n choices of g for f_{n-1}, and each choice has $n-1$ candidates for k. Moreover, among these $n-1$ candidates, $n-2$ comparisons are performed to acquire the minimum value for each f_{n-1}. To obtain the minimum makespan among these f_{n-1} needs $(n-1)$ comparisons. Therefore, the total number of additions required is:

$$= 2n(n-1)\sum_{i=1}^{n-2} i \cdot C_i^{n-2} + n(n-1) + n^2$$

$$= 2n(n-1)(n-2)\sum_{j=0}^{n-3} C_j^{n-3}) + 2n^2 - n$$

$$= 2n(n-1)(n-2)2^{n-3} + 2n^2 - n$$

$$\approx n(n-1)(n-2)2^{n-2}.$$

The total number of comparisons required is:

$$= n(n-1)(2\sum_{i=1}^{n-2} i \cdot C_i^{n-2} - \sum_{i=1}^{n-2} C_i^{n-2}) + n(n-1) + n(2n-3) + n - 1$$

$$= n(n-1)(2(n-2)2^{n-3} - 2^{n-2} + 1) + 3n^2 - 3n - 1$$

$$= 2n(n-1)(n-2)2^{n-3} + n(n-1)(2^{n-2}) + 4n^2 - 4n - 1$$

$$\approx n(n-1)(n-2)2^{n-2}.$$

Thus, the computational effort for this dynamic programming algorithm is $O(n^3 2^{n-2})$. This analysis of computational effort shows the exponential time complexity of the algorithm, which confirms with NP-hardness of the problem proven in Section 4.2. When computing a value of the optimal value function f_i, it is necessary to store several values of the function f_{i-1} in the previous stage. Given n is 15, for example, to calculate the values of f_{10} in stage 10, the total number of f_9 has to be computed and stored in the prior stage is 150,150 ($n(n-1)C_i^{n-2}$). Hence, the storage space required for the algorithm would be the practical restriction for solving the problem in a large scale.

A Numerical Example

A numerical example is presented to illustrate the proposed dynamic programming algorithm. In this example, 4 jobs need to be processed in a T-line machining center with one CNC machine, and one L/U station. Table 4.2 shows the loading, processing, and unloading times for these jobs.

Table 4.2: Job data for a one-machine T-line machining center

Job	l_j	p_j	u_j
1	3	7	2
2	4	6	3
3	6	10	4
4	1	3	3

- Boundary Condition:

$f_0(\varnothing,2,1) = 4 + \max\{6,3\} = 10.$
$f_0(\varnothing,3,1) = 6 + \max\{10,3\} = 16.$
$f_0(\varnothing,4,1) = 1 + \max\{3,3\} = 4.$

$f_0(\varnothing,1,2) = 3 + \max\{7,4\} = 10.$
$f_0(\varnothing,3,2) = 6 + \max\{10,4\} = 16.$
$f_0(\varnothing,4,2) = 1 + \max\{3,4\} = 5.$

$f_0(\varnothing,1,3) = 3 + \max\{7,6\} = 10.$
$f_0(\varnothing,2,3) = 4 + \max\{6,6\} = 10.$
$f_0(\varnothing,4,3) = 1 + \max\{3,6\} = 7.$

$f_0(\varnothing,1,4) = 3 + \max\{7,1\} = 10.$
$f_0(\varnothing,2,4) = 4 + \max\{6,1\} = 10.$
$f_0(\varnothing,3,4) = 6 + \max\{10,1\} = 16.$

- *i* = 1:

$f_1(2,3,1) = f_0(\varnothing,2,3) + \max\{10,3+3\} = 10 + 10 = 20.$
$f_1(2,4,1) = f_0(\varnothing,2,4) + \max\{3,3+3\} = 10 + 6 = 16.$
$f_1(3,2,1) = f_0(\varnothing,3,2) + \max\{6,3+4\} = 16 + 7 = 23.$
$f_1(3,4,1) = f_0(\varnothing,3,4) + \max\{3,3+4\} = 16 + 7 = 23.$
$f_1(4,2,1) = f_0(\varnothing,4,2) + \max\{6,3+3\} = 5 + 6 = 11.$
$f_1(4,3,1) = f_0(\varnothing,4,3) + \max\{10,3+3\} = 7 + 10 = 17.$

$f_1(1,3,2) = f_0(\varnothing,1,3) + \max\{10,4+2\} = 10 + 10 = 20.$
$f_1(1,4,2) = f_0(\varnothing,1,4) + \max\{3,4+2\} = 10 + 6 = 16.$
$f_1(3,1,2) = f_0(\varnothing,3,1) + \max\{7,4+4\} = 16 + 8 = 24.$
$f_1(3,4,2) = f_0(\varnothing,3,4) + \max\{3,4+4\} = 16 + 8 = 24.$
$f_1(4,1,2) = f_0(\varnothing,4,1) + \max\{7,4+3\} = 4 + 7 = 11.$
$f_1(4,3,2) = f_0(\varnothing,4,3) + \max\{10,4+3\} = 7 + 10 = 17.$

$f_1(1,2,3) = f_0(\varnothing,1,2) + \max\{6,6+2\} = 10+8 = 18.$
$f_1(1,4,3) = f_0(\varnothing,1,4) + \max\{3,6+2\} = 10+8 = 18.$
$f_1(2,1,3) = f_0(\varnothing,2,1) + \max\{7,6+3\} = 10+9 = 19.$
$f_1(2,4,3) = f_0(\varnothing,2,4) + \max\{3,6+3\} = 10+9 = 19.$
$f_1(4,1,3) = f_0(\varnothing,4,1) + \max\{7,6+3\} = 4+9 = 13.$
$f_1(4,2,3) = f_0(\varnothing,4,2) + \max\{6,6+3\} = 5+9 = 14.$

$f_1(1,2,4) = f_0(\varnothing,1,2) + \max\{6,1+2\} = 10+6 = 16.$
$f_1(1,3,4) = f_0(\varnothing,1,3) + \max\{10,1+2\} = 10+10 = 20.$
$f_1(2,1,4) = f_0(\varnothing,2,1) + \max\{7,1+3\} = 10+7 = 17.$
$f_1(2,3,4) = f_0(\varnothing,2,3) + \max\{10,1+3\} = 10+10 = 20.$
$f_1(3,1,4) = f_0(\varnothing,3,1) + \max\{7,1+4\} = 16+7 = 23.$
$f_1(3,2,4) = f_0(\varnothing,3,2) + \max\{6,1+4\} = 16+6 = 22.$

- $i = 2$:

$f_2(\{2,3\},4,1) = \min\{f_1(2,3,4) + \max\{3,3+4\},\ f_1(3,2,4) + \max\{3,3+3\}\} = 27.$
$f_2(\{2,4\},3,1) = \min\{f_1(2,4,3) + \max\{10,3+3\},\ f_1(4,2,3) + \max\{10,3+3\}\} = 24.$
$f_2(\{3,4\},2,1) = \min\{f_1(3,4,2) + \max\{6,3+3\},\ f_1(4,3,2) + \max\{6,3+4\}\} = 24.$

$f_2(\{1,3\},4,2) = \min\{f_1(1,3,4) + \max\{3,4+4\},\ f_1(3,1,4) + \max\{3,4+2\}\} = 28.$
$f_2(\{1,4\},3,2) = \min\{f_1(1,4,3) + \max\{10,4+3\},\ f_1(4,1,3) + \max\{10,4+2\}\} = 23.$
$f_2(\{3,4\},1,2) = \min\{f_1(3,4,1) + \max\{7,4+3\},\ f_1(4,3,1) + \max\{7,4+4\}\} = 25.$

$f_2(\{1,2\},4,3) = \min\{f_1(1,2,4) + \max\{3,6+4\},\ f_1(2,1,4) + \max\{3,6+2\}\} = 25.$
$f_2(\{1,4\},2,3) = \min\{f_1(1,4,2) + \max\{6,6+3\},\ f_1(4,1,2) + \max\{6,6+2\}\} = 19.$
$f_2(\{2,4\},1,3) = \min\{f_1(2,4,1) + \max\{7,6+3\},\ f_1(4,2,1) + \max\{7,6+3\}\} = 20.$

$f_2(\{1,2\},3,4) = \min\{f_1(1,2,3) + \max\{10,1+3\},\ f_1(2,1,3) + \max\{10,1+2\}\} = 28.$
$f_2(\{1,3\},2,4) = \min\{f_1(1,3,2) + \max\{6,1+4\},\ f_1(3,1,2) + \max\{6,1+2\}\} = 26.$
$f_2(\{2,3\},1,4) = \min\{f_1(2,3,1) + \max\{7,1+4\},\ f_1(3,2,1) + \max\{7,1+3\}\} = 27.$

- Answer:

$\min\{f_3(\{1,2,3\},4,\varnothing),\ f_3(\{1,2,4\},3,\varnothing),\ f_3(\{1,3,4\},2,\varnothing),\ f_3(\{2,3,4\},1,\varnothing)\} = \max\{32,33,32,33\} = 32$

where

$$
\begin{aligned}
f_3(\{1,2,3\},4,\varnothing) &= \min\{ f_2(\{1,2\},3,4)+\max\{3,4\},\ f_2(\{1,3\},2,4)+\max\{3,3\},\ f_2(\{2,3\},1,4)+\max\{3,2\}\}+3 \\
&= \max\{28+4, 26+3, 27+3\}+3=32. \\
f_3(\{1,2,4\},3,\varnothing) &= \min\{ f_2(\{1,2\},4,3)+\max\{10,3\},\ f_2(\{1,4\},2,3)+\max\{10,3\},\ f_2(\{2,4\},1,3)+\max\{10,2\}\}+4 \\
&= \max\{25+10, 19+10, 20+10\}+4=33. \\
f_3(\{1,3,4\},2,\varnothing) &= \min\{ f_2(\{1,3\},4,2)+\max\{6,3\},\ f_2(\{1,4\},3,2)+\max\{6,4\},\ f_2(\{3,4\},1,2)+\max\{6,2\}\}+3 \\
&= \max\{28+6, 23+6, 25+6\}+3=32. \\
f_3(\{2,3,4\},1,\varnothing) &= \min\{ f_2(\{2,3\},4,1)+\max\{7,3\},\ f_2(\{2,4\},3,1)+\max\{7,4\},\ f_2(\{3,4\},2,1)+\max\{7,3\}\}+2 \\
&= \max\{27+7, 24+7, 24+7\}+2=33.
\end{aligned}
$$

Thus, the minimum makespan is 32, and the optimal sequence is $1-3-2-4$ or $4-1-3-2$.

4.5 Two-Phase Heuristic Algorithm for One CNC Machine

Although the problem can be solved by the proposed dynamic programming algorithm, the computational effort of the algorithm is not polynomial. In addition, since the complexity of the problem is proven as NP-hard in the strong sense, it is not likely to develop polynomial time algorithms for this type of problems. Thus, developing a heuristic algorithm is a feasible method to address the problem in a large scale.

Heuristic algorithms have been developed to provide good or near-optimal solutions in a reasonable CPU time. They are typically applied to large-size problems with limited computational effort. The heuristic algorithms for the scheduling problems are categorized into two types: constructive heuristics and improvement heuristics. The constructive heuristics help to build a feasible schedule from scratch, and the improvement heuristics attempt to improve the quality of the solution iteratively from the given schedule. Framinan *et al.* (2004) define a constructive heuristic as building a solution in a recursive manner by trying to insert one or more unscheduled job into one or more positions of a partial schedule until all jobs are sequenced.

Contrary to constructive heuristics, an improvement heuristic starts from an initial sequence and tries to improve the solution through modifications to that sequence. The improvement procedure is repeatedly applied while improvements are gained. Local searches and metaheuristics such as Tabu search, simulated annealing, genetic algorithms, and ant colony optimization are categorized as improvement heuristics (Framinan *et al.* 2004; Hejazi and Saghafian 2005). Nowadays, most of the heuristic algorithms proposed in the literature regarding job scheduling are to combine constructive and improvement heuristics as two-phase algorithms.

The proposed heuristics algorithm in this research also contains two following stages: (1) the constructive stage – in this phase an initial sequence is formed and an initial makespan is determined, and (2) the improvement stage, a search for a better solution based on the initial seed. Two heuristic algorithms in the constructive stage are proposed to generate an initial seed. In the improvement stage, a modified neighborhood search algorithm is developed to explore solution spaces and to find improvements in the makespan. These algorithms are presented and discussed in the following section.

Constructive Stage

Two constructive heuristics are proposed to form an initial sequence for the one-machine problem. One heuristic algorithm is to choose a suitable job based on a given processing time and to append it to the current sequence. The other is to insert a given job in every position of the current sequence to find a best position for that job. The details of these two algorithms are discussed as below.

Constructive Algorithm – Selection (CAS)

Recall that the makespan for the problem $F_1 \mid synmv, re-LU \mid C_{\max}$ is to sum up time of each cycle, $\sum_{i=1}^{n+2} C_i$. The time period of each cycle is equal to the maximum operation time among these times at the L/U station and on the CNC machine. If the summation of processing times on the CNC machine is greater than the summation of loading and unloading times at the L/U station, then the summation of processing times can be regarded as a lower bound of the solution. This property can be shown as in Lemma 1.

Lemma 4.1. The value, $\min\limits_{i=1,\ldots,n} l_i + \sum_{j=1}^{n} p_j + \min\limits_{i=1,\ldots,n} u_i$, provides a lower bound for the makespan problem in a T-line machining center with one CNC machine.

Proof: According to Equations (4.1) to (4.3), the makespan of a given sequence, say σ, is represented as $MS_\sigma = l_{[1]} + \max\{p_{[1]},\ l_{[2]}\} + \max\{p_{[2]},\ u_{[1]} + l_{[3]}\} + \ \ldots\ + \max\{p_{[n-1]},\ u_{[n-2]} +\ l_{[n]}\} + \max\{p_{[n]},\ u_{[n-1]}\} + u_{[n]}$. $\min\limits_{i=1,\ldots,n} l_i$ and $\min\limits_{i=1,\ldots,n} u_i$ are the lower bounds for $l_{[1]}$ and $u_{[n]}$, respectively. In addition, by neglecting the loading and unloading from cycles 2 to *n*+1. Thus,

$$MS_\sigma \geq l_{[1]} + p_{[1]} + p_{[2]} + \ldots + p_{[n]} + u_{[n]} \geq \min_{i=1,\ldots,n} l_i + \sum_{j=1}^{n} p_j + \min_{i=1,\ldots,n} u_i \qquad \square$$

Furthermore, from the above lemma we can conclude that if the minimum processing time on the CNC machine is greater than the sum of the largest loading and unloading time, the optimal solution is equal to the lower bound. The property can be represented as follows:

$$\min_{j=1,\ldots,n} p_j \geq \max_{i=1,\ldots,n} l_i + \max_{i=1,\ldots,n} u_i \Rightarrow \text{the optimal makespan: } \min_{i=1,\ldots,n} l_i + \sum_{j=1}^{n} p_j + \min_{i=1,\ldots,n} u_i .$$

In other words, the processing times can serve as the basic elements of comparison with different possible combinations of loading and unloading times at the L/U station. Therefore, the main idea of constructing the initial job sequence is to find the summation of a pair of loading and unloading times that is closest to a selected processing time. The heuristic is named as Constructive Algorithm – Selection (CAS), and the steps associated with the algorithm are stated below.

CAS Algorithm

N is a set containing all jobs; *N* = {1, 2, … , *n*}. *S* is a set containing jobs which have been already sequenced. *R* is a set containing jobs which have not been sequenced yet. *MS* is a variable to record the current makespan.

Step 1. Let *R* = *N* and *S* = ∅.

Step 2. Select a job which has the largest processing time from set *R*; name the job as *j*; $j \in arg\ p_j = \max\{p_i : i \in R\}$. Next, find a pair of jobs (*f*, *b*) where $\{f, b\} \subseteq R \setminus \{j\}$. The summation of loading and unloading times of these two jobs should be closest to the processing time of job *j*; $(f, b) \in arg\ \min\{| p_j - (l_b + u_f) | : (f, b) \in R \setminus \{j\}\}$. The tie break rule is to select the pair with negative value of $p_j - (l_b + u_f)$. Then, job *f* is sequenced prior to job *j*, and job *b* is sequenced after job *j*. The makespan of the current sequence can be calculated as $MS = l_f + \max\{p_f, l_j\} + \max\{p_j, u_f + l_b\} + \max\{p_b, u_j\} + u_b$. Let the index *k* = *j*, $S = \{f, j, b\}$, and $R = N \setminus S$.

Step 3. Select a job with a larger processing time between job *f* and *b*.

Step 3.1. If $p_f \geq p_b$, then select a job, say *i*, from set *R* with the unloading time which is closest to $\max\{p_f - l_j, 0\}$; $i \subseteq R$. The tie break rule is $\min\{u_i - \max\{p_f - l_j, 0\}\}$. Append job *i* at the beginning of the current sequence, and the makespan becomes $MS + l_i + \max\{p_i - l_f, 0\} + (\max\{u_i + l_j - p_f, 0\} \mid p_f \geq l_j) + (u_i \mid p_f < l_j)$. Set *j* = *f* and *f* = *i*.

Step 3.2. If $p_f < p_b$, then select a job, say *i*, from set *R* with loading time which is closest to $\max\{p_b - u_k, 0\}$; $i \subseteq R$. The tie break rule is $\min\{l_i - \max\{p_b - u_k, 0\}\}$. Add job *i* at the end of the current sequence, and the makespan becomes $MS = MS + u_i + \max\{p_i - b_u, 0\} + (\max\{u_k + l_i - p_b, 0\} \mid p_b \geq u_k) + (l_i \mid p_b < u_k)$. Set *k*= *b* and *b*= *i*.

Step 3.3. Let $S = S \cup \{i\}$, and $R = R \setminus \{i\}$. If $R \neq \varnothing$, go to step 3. Otherwise, go to step 4.

Step 4. Output the job sequence and the makespan.

A Numerical Example

A numerical example is presented to demonstrate the procedure of CAS. Assume there are five jobs. Table 4.3 shows the job data for the example.

Table 4.3: Job data for the example of CAS

Job	l_j	p_j	u_j
1	4	7	4
2	3	9	3
3	5	3	3
4	2	5	1
5	2	7	5

Iteration 1:

Step 2. The largest processing time is 9, and the corresponding job is 2. Find a pair of jobs (*f*, *b*) where the summation of loading and unloading times of the two jobs is closest to 9. The pair of jobs selected is job 1 and job 3. The current job sequence is $1-2-3$.

Step 3. Job 1 has a larger processing time than job 3.

Step 3.1 Select job *i* with an unloading time close to $\max\{p_f - l_j, 0\} = \max\{7-3, 0\} = 4$. Job 5 is selected. The current job sequence is $5-1-2-3$.

Iteration 2:

Step 3. Job 5 has a larger processing time than job 3.

Step 3.1 Sequence the remaining job 4 in the front of the sequence.

Step 4. The job sequence is $4-5-1-2-3$, and the makespan is 37. (It is also optimal.)

Constructive Algorithm – Insertion (CAI)

Similar to the idea of the first constructive algorithm, if the cycle times for cycle 2 to cycle *n*+1 are identified by loading and unloading times, the summation of these loading and unloading times will provide a lower bound to a makespan of any sequence. This property is stated as in Lemma 4.2.

Lemma 4.2. The value, $\sum_{j=1}^{n}(l_j + u_j)$, provides a lower bound for the makespan problem in a T-line machining center with one CNC machine.

Proof: According to Equations (4.1) to (4.3), the makespan of a given sequence, say σ, is represented as $MS_\sigma = l_{[1]} + \max\{p_{[1]},\ l_{[2]}\} + \max\{p_{[2]},\ u_{[1]} + l_{[3]}\} + \ \dots\ + \max\{p_{[n-1]},\ u_{[n-2]} + l_{[n]}\} + \max\{p_{[n]},\ u_{[n-1]}\} + u_{[n]}$. Neglect the processing times in these cycles 2 to n+1. Thus,

$$MS_\sigma \geq l_{[1]} + l_{[2]} + (l_{[3]} + u_{[1]}) \dots + (l_{[n]} + u_{[n-2]}) + u_{[n-1]} + u_{[n]} = \sum_{j=1}^{n} (l_j + u_j)$$ □

Furthermore, from the above lemma we can derive that if the maximum processing time on the CNC machine is less than the smallest loading and unloading time, the optimal solution is equal to the lower bound. The property can be represented as follows:

$$\max_{j=1,\dots,n} p_j \leq \min_{i=1,\dots,n} l_i \text{ and } \max_{j=1,\dots,n} p_j \leq \min_{i=1,\dots,n} u_i \Rightarrow \text{the optimal makespan: } \sum_{j=1}^{n} (l_j + u_j).$$

In addition, a new lower bound, which provides a tighter bound for any sequence, can be derived from Lemma 4.1 and Lemma 4.2.

Theorem 4.3. The value, $\max\{\min_{i=1,\dots,n} l_i + \sum_{j=1}^{n} p_j + \min_{i=1,\dots,n} u_i,\ \sum_{j=1}^{n} (l_j + u_j)\}$, provides a lower bound for the makespan problem in a T-line machining center with one CNC machine.

Proof: Derived from Lemmas 4.1 and 4.2 directly. □

Therefore, the main idea of the constructive algorithm is to sequence a job based on its loading and unloading time. Each job is assigned two weights: one weight is determined by its loading time, the other weight is determined by its unloading time. A job with larger loading or unloading times is assigned larger weights, and a job with the large sum of these two weights will be sequenced first. When a job is selected to be inserted into the current sequence, this job will be inserted in every position of the sequence to obtain a corresponding makespan. For a given subsequence with k jobs, there are k+1 inserting positions: the beginning and the end positions of the subsequence and the positions between two consecutive jobs in the subsequence. After all attempts in each position, the job will be sequenced in the position where the minimum makespan is obtained. The processes are repeated until all jobs are sequenced. The procedure of this algorithm, Constructive Algorithm – Insertion (CAI), is stated as follows.

CAI Algorithm

N is a set containing all jobs; $N = \{1, 2, \ldots, n\}$. S is a set containing jobs which have been sequenced, and R is a set containing jobs which are not sequenced. Each job has two weights which are assigned according to its loading and unloading times. The weights corresponding to loading and unloading times of job j are w_{lj} and w_{uj}, respectively. MS is a variable to record the current makespan.

Step 1. Let $MS = 0$, $R = N$ and $S = \emptyset$.

Step 2. Sort jobs based on loading times in ascending order, and assign a weight value to each job. The weight value for the first job is 1, for the second job is 2 and for the last job is n. If jobs have the same loading times, the weights assigned to these jobs are the same. For example, k jobs have the same loading time, and the weight value assigned to these jobs is w. However, the weight assigned to next job will be w+k. Similarly, the job with the smallest unloading time has a weight value equal to 1, and the job with largest unloading time has a weight value equal to n.

Step 3. Select a job (denoted as j) from set R with the largest value of $w_{lj} + w_{uj}$.

Step 4. Insert job j in position i of the current sequence where $i = 1, \ldots, |S| + 1$. The corresponding makespan of inserting job j in position i (denoted as MS_{ij}) can be calculated as follows:

Step 4.1. $MS_{ij} = MS + \max\{p_{[i-1]}, u_{[i-2]} + l_j\} + \max\{p_j, u_{[i-1]} + l_{[i]}\} + \max\{p_{[i]}, u_j + l_{[i+1]}\}$

$- \max\{p_{[i-1]}, u_{[i-2]} + l_{[i]}\} - \max\{p_{[i]}, u_{[i-1]} + l_{[i+1]}\}$

where $l_{[|S|+1]} = l_{[|S|+2]} = p_{[0]} = p_{[|S|+1]} = u_{[-1]} = u_{[0]} = 0$.

Step 4.2. Select position i with the minimum makespan, and insert job j in that position. Let $MS = \min\{MS_{ij}\}$, $S = S \cup \{j\}$ and $R = R \setminus \{j\}$. If $R \neq \varnothing$, go to step 3. Otherwise, go to step 5.

Step 5. Output the job sequence and the makespan.

A Numerical Example

A numerical example is presented to demonstrate the procedure of CAI. Table 4.4 shows the job data for the example.

Table 4.4: Job data for the example of CAI

Job	l_j	w_{lj}	p_j	u_j	w_{uj}	$w_{lj} + w_{uj}$
1	4	4	7	4	4	8
2	3	3	9	3	2	5
3	5	5	3	3	2	7
4	2	1	5	1	1	2
5	2	1	7	5	5	6

Iteration 1:

Step 3. R = {1, 3, 5, 2, 4}. Job 1 is selected.

Step 4. Insert job 1 to the current sequence, and the current makespan is 15. S = {1}.

Iteration 2:

Step 3. R = {3, 5, 2, 4}. Job 3 is selected.

Step 4. Insert job 3 to the current sequence.

Step 4.1. $i = 1$, $MS = 20$.

$i = 2$, $MS = 18$.

Step 4.2.Insert job 3 in position 2. $S = \{1, 3\}$ and $MS = 19$.

Iteration 3:

Step 3. $R = \{5, 2, 4\}$. Job 5 is selected.

Step 4. Insert job 5 to the current sequence.

Step 4.1. $i = 1$, $MS = 26$.

$i = 2$, $MS = 28$.

$i = 3$, $MS = 29$.

Step 4.2.Insert job 3 in position 1. $S = \{5, 1, 3\}$ and $MS = 26$.

Iteration 4:

Step 3. $R = \{2, 4\}$. Job 2 is selected.

Step 4. Insert job 2 to the current sequence.

Step 4.1. $i = 1$, $MS = 36$.

$i = 2$, $MS = 33$.

$i = 3$, $MS = 32$.

$i = 4$, $MS = 38$.

Step 4.2.Insert job 3 in position 1. $S = \{5, 1, 2, 3\}$ and $MS = 32$.

Iteration 5:

Step 3. $R = \{4\}$. Job 4 is selected.

Step 4. Insert job 4 to the current sequence.

Step 4.1. $i = 1$, $MS = 37$.

i = 2, MS = 40.

i = 3, MS = 38.

i = 4, MS = 40.

i = 5, MS = 37.

Step 4.2.Insert job 4 in position 1 or 5. S = {4, 5, 1, 2, 3} or {5, 1, 2, 3, 4} and MS = 37.

Step 5. The job sequence is $4-5-1-2-3$ or $5-1-2-3-4$, and the makespan is 37. (It is also optimal.)

Improvement Stage

After generating a seed, a better solution or sequence will be searched for improving the current makespan. Typically, neighborhood solutions of the seed are generated and explored. Then the sequence with the smallest makespan among these neighborhood sequences is selected as the seed for next iteration of the improvement stage. The procedure does not terminate until a further improved sequence cannot be found. The technique of the search algorithm is referred to as Neighborhood Search. It is important to determine the method of generating neighborhood solutions in the search algorithm, because the more possible candidate solutions are explored, the better improvements can be obtained. One of mechanisms to generate neighborhood sequences is known as adjacent pairwise interchange, which is to switch two adjacent jobs. In the proposed algorithm, not only the adjacent pairwise interchange is adopted, but also pairwise interchange of any two jobs is considered.

One of the weaknesses of the neighborhood search algorithm is that the current solution may be trapped in a local optimum so that no better neighbor can be found with respect to the current seed. In order to escape from a local optimum, a mechanism to increase the diversification of the search region is incorporated into the neighborhood search algorithm. If no more improvements can be found in the neighborhood region of the current seed, a neighborhood sequence with the identical makespan as the current seed is selected as a new seed. If there are more than one neighborhood sequences with the same makespan as the current seed, one sequence is randomly chosen from them. In the next iteration, the unexplored region of the new seed can be searched for further improvement. Therefore, these two mechanisms, the pairwise interchange and the escape from a local optimum, comprise the basic structure of the algorithm in the improvement stage. The modified neighborhood search algorithm is explained in detail in the rest of this section.

Algorithm of Modified Neighborhood Search

Notations:

B: the current best sequence.

MS(*B*): the makespan of the sequence *B*.

MS(*S*): the makespan of the sequence *S*.

MS(*S'*): the makespan of the sequence *S'*.

R: the set containing the sequences with identical makespan as the seed.

Counter: a counter to record the number of iterations that has been performed.

Step 1. Let the sequence obtained from the constructed stage be the initial seed *S*.

Step 2. Initialize *MS*(*B*) as a very large value, and perform adjacent pairwise interchange $i = 1$.

Step 2.1. Generate *S'* by swapping the positions of job *i* and job *i*+1 in *S*. The makespan can be obtained as the following formulation.

$$MS(S') = MS(S) - \sum_{j=i}^{i+3} \max\{p_{[j-1]}, u_{[j-2]} + l_{[j]}\} + \max\{p_{[i-1]}, u_{[i-2]} + l_{[i+1]}\}$$

$$+ \max\{p_{[i+1]}, u_{[i-1]} + l_{[i]}\} + \max\{p_{[i]}, u_{[i+1]} + l_{[i+2]}\} + \max\{p_{[i+2]}, u_{[i]} + l_{[i+3]}\}$$

where $l_{[n+1]} = l_{[n+2]} = p_{[0]} = p_{[n+1]} = u_{[-1]} = u_{[0]} = 0$.

Step 2.2. If *MS*(*S'*) < *MS*(*B*), then $B \leftarrow S'$ and *MS*(*B*) = *MS*(*S'*).

Step 2.3. If *MS*(*S'*) = *MS*(*S*), then $R = R \cup S'$.

Step 2.4. $i = i+1$ and go to Step 2.1 until $i = n-1$.

Step 3. If *MS*(*B*) < *MS*(*S*) then $S \leftarrow B$, $R = \varnothing$, $Counter = 1$, and go to Step 2. Otherwise go to step 4.

Step 4. Initialize *MS*(*B*) as a very large value, and perform pairwise interchange from $i = 1$.

Step 4.1. Generate *S'* by swapping the job positions of job *i* and job *j* where $j \geq i+2$ in *S*, and the makespan can be obtained as the following formulation.

If $j = i+2$, the makespan can be calculated as below:

$$MS(S') = MS(S) - \sum_{j=i}^{i+4} \max\{p_{[j-1]}, u_{[j-2]} + l_{[j]}\}$$

$$+ \max\{p_{[i-1]}, u_{[i-2]} + l_{[j]}\} + \max\{p_{[j]}, u_{[i+1]} + l_{[i+1]}\} + \max\{p_{[i+1]}, u_{[j]} + l_{[i]}\}$$

$$+ \max\{p_{[i]}, u_{[j-1]} + l_{[j+1]}\} + \max\{p_{[j+1]}, u_{[i]} + l_{[j+2]}\}$$

where $l_{[n+1]} = l_{[n+2]} = p_{[0]} = p_{[n+1]} = u_{[-1]} = u_{[0]} = 0$.

Else

$$MS(S') = MS(S) - \sum_{k=i}^{i+2} \max\{p_{[k-1]}, u_{[k-2]} + l_{[k]}\} + \sum_{k=j}^{j+2} \max\{p_{[k-1]}, u_{[k-2]} + l_{[k]}\}$$

$$+\max\{p_{[i-1]}, u_{[i-2]} + l_{[j]}\} + \max\{p_{[j]}, u_{[i-1]} + l_{[i+1]}\} + \max\{p_{[i+1]}, u_{[j]} + l_{[i+2]}\}$$

$$+\max\{p_{[j-1]}, u_{[j-2]} + l_{[i]}\} + \max\{p_{[i]}, u_{[j-1]} + l_{[j+1]}\} + \max\{p_{[j+1]}, u_{[i]} + l_{[j+2]}\}$$

where $l_{[n+1]} = l_{[n+2]} = p_{[0]} = p_{[n+1]} = u_{[-1]} = u_{[0]} = 0$.

Step 4.2. If *MS*(*S'*) < *MS*(*B*), then $B \leftarrow S'$, *MS*(*B*) = *MS*(*S'*) .

Step 4.3. If *MS*(*S'*) = *MS*(*S*), then $R = R \cup S'$.

Step 4.4. *j* = *j*+1 and go to Step 4.1 until *j* = *n*.

Step 5. If MS(*B*) < MS(*S*), then $S \leftarrow B$, $R = \varnothing$, $Counter = 1$, and go to Step 2. Otherwise *i* = *i*+1 and go to step 4 until $i = n-2$.

Step 6. If *Counter* < 10000, then randomly select a sequence from *R* as a new seed *S*. $R = \varnothing$, $Counter = Counter + 1$, and go to Step 2. Otherwise go to step 7.

Step 7. Output the final sequence and the makespan.

Starting the neighborhood search requires an initial sequence as seed *S*. This seed is generated from the algorithm in the constructive stage. First, a series of adjacent pairwise interchanges is performed on the seed to generate a list of new sequences. The adjacent pairwise interchange is to swap the positions of two adjacent jobs in a sequence. The interchange starts from the first job in a sequence and continues until the last job. After swapping the positions of the first two jobs, a new sequence *S'* is generated, and the makespan *MS*(*S'*) of the sequence is computed.

A second new sequence is generated by exchanging the positions of the second and third jobs. If the makespan of the second sequence is smaller than the makespan of the first one, the second sequence will serve as the current best sequence. The procedure will be repeated until the

$n-1^{th}$ sequence is generated and compared with the current best sequence. Hence, the current best sequence is the one among the seed's neighbors with the smallest makespan. The current best sequence will become a new seed for the next iteration.

If no better sequence could be obtained, another pairwise interchange mechanism will be applied. This mechanism will swap the job in position *i* and the job in position *j* where *i* is from 1 to $n-2$, and *j* is from *i*+2 to *n*. The procedure is similar to the procedure of the adjacent pairwise interchange. When *i* is equal to 1, for example, $n-2$ sequences are generated and the sequence with the minimum makespan will serve as a seed for next iteration. If no better sequence is obtained, then *i* increases by 1 and $n-i-1$ sequences are generated. The procedure isn't terminated until either a better solution is found or $i=n-1$. The total number of possible solutions in the neighborhood region of the seed explored is $\frac{n(n-1)}{2}$.

If no sequence with an improved makespan can be obtained after performing these two interchange schemes, the current seed could be considered as a local optimum in terms of the scheme of the pairwise interchange. Therefore, a remedial method should be adopted to increase the diversification of the search algorithm and avoid the solution trapping in a local optimum. The method incorporated in the algorithm of the improvement stage is to randomly select a sequence which has the identical makespan as the current solution as a new seed. Then the neighborhood search is applied to the new seed to search a better sequence. There is a counter to record the number of the random selections is performed. However, once a neighbor sequence with a better makespan is obtained, the counter is reset to 1. The procedure of random selection is executed repeatedly until the counter reaches a predefined value (i.e., 10000). In this condition, the whole improvement stage is terminated, and the current sequence and the makespan are reported.

4.6 Computational Results

In order to evaluate the performance of the proposed heuristic algorithms, a series of experiments is conducted. The solutions generated by the two developed algorithms are compared with the optimal solutions obtained by the dynamic programming algorithm in Section 4.4. Since no sample problems have been found in the literature and no real data from the machining center is available, all of the testing data are randomly created in this research.

In order to assure the robustness of the experiments, three testing scenarios are performed: (1) the expected value of processing times is equal to the expected value of the sum of loading and unloading times, (2) the expected value of processing times is greater than the expected value of the sum of loading and unloading times, and (3) the expected value of the sum of loading and unloading times is greater than the expected value of processing times. Typically, the operation of loading (i.e., loading jobs and adjust fixtures) is more complicated than the operation of unloading. Therefore, the expected value of loading times is set to be larger than the expected value of unloading times. In addition, for each case the small, medium and large size problems are also considered. The makespan values and CPU times are recorded for all experiments. The rules to generate the testing data are summarized in Table 4.5.

Table 4.5: Experimental design and data generating rules for $F_1 \mid synmv, re-LU \mid C_{\max}$

	Small size	Medium size	Large size
Number of jobs (n)	10/15	19	40
Scenario I (7, 11, 3)	(l_j, p_j, u_j) = (U(1,7), U(1, 11), U(1, 3))		
Scenario II (7, 15, 3)	(l_j, p_j, u_j) = (U(1,7), U(1, 15), U(1, 3))		
Scenario III (10, 11, 4)	(l_j, p_j, u_j) = (U(1,10), U(1, 11), U(1, 4))		

*U denotes the discrete distribution and all operation times are integer.

Two constructive algorithms, CAS and CAI, are developed and each of them is integrated with the modified neighborhood search algorithm as one two-phase algorithm. Thus, two two-phase algorithms are implemented and they are named as CAS_M and CAI_M, respectively. These two heuristic algorithms are coded by using Borland C++ 5.5 as well as the dynamic programming algorithm. Ten runs are executed for each scenario. These tests are run on a Pentium 1.40GHz PC with 1 GB RAM. For small-size problems, 10 and 15 jobs are tested. According to the computational analysis of the dynamic programming algorithm in Section 4.4, the maximum number of jobs in the problem that can be solved optimally by the dynamic programming algorithm is 19, due to the restriction of RAM size. Even increasing the memory size or the virtual memory size can't address the issue. That is the reason that the number of jobs in the medium size is set to 19.

Results

There are three scenarios in the four different size problems and 10 runs are executed for each case. Hence, 120 instances are tested for these two proposed heuristics. For the small and medium size problems, the optimal solutions can be obtained. In a large-size problem, however, it is impractical to obtain an optimal solution because the problem is strongly NP-hard. As a result, only a lower bound value is derived from Theorem 4.3 as a reference value to be compared with the solution obtained by heuristic algorithms. The average makespan on 10 runs for each case is summarized in Table 4.6. In addition, the percentage of the makespan generated by heuristics from the optimum and lower bound is measured by the relative error. Table 4.7 illustrates the average relative error from the optimum and lower bound.

Table 4.6. Summary of the average makespans obtained by the DP, heuristics, and LB

Scenario	*n*	DP		CAS_M			CAI_M			LB
		Optimum	Time	S1	S2	Time	S1	S2	Time	
I (7, 11, 3)	10	68.9	0.06	74.4	69.3	0.08	71.1	69.5	0.08	68.2
	15	95.1	4.39	103.3	95.7	0.21	98.3	95.8	0.20	93.1
	19	121.8	344.8	136.0	122.6	0.26	126.0	123.1	0.29	119.6
	40	–	–	280.7	253.2	0.92	261.3	252.7	1.12	247.1
II (7, 15, 3)	10	82.2	0.06	86.8	82.4	0.08	83.4	82.6	0.08	80.7
	15	131.3	4.37	137.6	131.5	0.20	134.1	131.5	0.21	129.4
	19	163.1	356.1	171.4	163.2	0.26	165.5	163.4	0.25	160.3
	40	–	–	333.1	314.7	1.03	321.7	314.8	1.02	310.4
III (10, 11, 4)	10	80.2	0.07	85.2	80.9	0.09	81.8	80.9	0.08	79.5
	15	124.0	4.41	131.8	124.6	0.21	126.6	124.4	0.20	123.5
	19	155.2	354.4	162.0	155.6	0.27	157.6	155.8	0.26	155.2
	40	–	–	335.9	323.0	0.86	323.2	323.0	0.83	322.1

(S1: the constructive stage; S2: the improvement stage)

Table 4.7: Summary of average relative errors from optimum and LB

Scenario	*n*	RE from optimum (%)				RE from LB (%)			
		CAS_M		CAI_M		CAS_M		CAI_M	
		S1	S2	S1	S2	S1	S2	S1	S2
I (7, 11, 3)	10	8.12	0.64	3.30	0.97	9.27	1.72	4.41	2.05
	15	8.68	0.63	3.33	0.74	11.02	2.79	5.55	2.90
	19	11.73	0.66	3.51	1.09	13.89	2.56	5.48	3.00
	40	–	–	–	–	13.82	2.50	5.80	2.29
II (7, 15, 3)	10	5.73	0.25	1.49	0.48	7.75	2.17	3.44	2.41
	15	5.10	0.18	2.25	0.17	6.88	1.85	3.98	1.85
	19	5.37	0.09	1.65	0.24	7.63	2.20	3.83	2.37
	40	–	–	–	–	7.45	1.43	3.71	1.46
III (10, 11, 4)	10	6.17	0.93	2.07	0.89	7.19	1.91	3.09	1.86
	15	6.38	0.53	2.18	0.35	6.86	0.99	2.65	0.81
	19	4.51	0.28	1.62	0.41	4.51	0.28	1.62	0.41
	40	–	–	–	–	4.29	0.29	0.36	0.29

For the small-size problems, optimal solutions can be obtained by the dynamic programming algorithm within 0.07 and 4.4 seconds for 10-job and 15-job cases, respectively. The execution times of these two heuristic algorithms are less than 0.2 seconds for all runs. The average relative errors from the optimal makespans are less than 1% for both CAS_M and CAI_M. However, the relative error for CAS in the constructive stage is up to 8.68% which is almost 3 times more than CAI.

When the number of jobs increases to 19, it requires 350 seconds for the dynamic programming algorithm to solve the problem on average. However, these two heuristics only require 0.3 seconds to obtain a solution, which are faster than the dynamic programming algorithm. Moreover, the relative errors from the optimal makespan are less than 1% for all scenarios, especially in the case of Scenario II and III.

For the large-size problems, because optimal solutions are unlikely to be obtained, the makespans generated by the heuristic algorithms are compared with the lower bound values. By observing the results in the small and medium size problems, the relative errors for the two heuristic algorithms from the lower bounds are within 3%. Hence, the lower bound provides a good reference compared with the optimal solution while the optimal solution is not available. In the worst case, the average relative errors from the lower bounds for CAS_M and CAI_M are 2.5% and 2.29%, respectively. This implies that the average relative error from the optimal makespan will be less than 2.5%. In addition, the CPU times required by these two algorithms are around 1 second. Hence, the time constraint is not an issue for executing the proposed algorithms.

Conclusions

Through these testing scenarios, the relative errors of the makespans obtained by these two proposed algorithms (CAS_M and CAI_M) compared with the optimal solutions or lower bound values are within 3% on average, with the worse case being 5%. Moreover, optimal sequences can be found in most cases when processing times are greater or less than the sum of the loading and unloading times, especially when the number of jobs is large. With respect to the constructive stage, solutions constructed by CAI are much better than those by CAS. In addition, the modified neighborhood search algorithm indeed provides significant improvements on the initial sequence. Regarding the computational effort, even for the large-sized problem, the average CPU time required by the proposed algorithms is no more than 2 seconds. Overall, because both of the proposed algorithms determine optimal or near-optimal solutions rapidly, we can conclude that the proposed two-phase heuristics are very suitable for solving the scheduling problem of a T-line machining center with one CNC machine.

Furthermore, the lower bound derived from Theorem 4.3 is a tight bound for the optimal solution, and provides an insightful reference when the optimum is unavailable. Observing the results of scenario II in the large-size problems, when the summation of processing times ($\sum_{j=1}^{n} p_j$) dominates the summation of loading and unloading time ($\sum_{j=1}^{n}(l_j + u_j)$), the lower bound value ($\min_{i=1,\ldots,n} l_i + \sum_{j=1}^{n} p_j + \min_{i=1,\ldots,n} u_i$) is closer to the optimal makespan. On the other hand, in the scenario III, if $\sum_{j=1}^{n}(l_j + u_j)$ is much greater than $\sum_{j=1}^{n} p_j$, then the difference between the lower bound $\sum_{j=1}^{n}(l_j + u_j)$ and the optimal value becomes smaller.

4.7 Concluding Summary

In summary, the scheduling problem in a one-machine T-line machining center has been studied and several contributions have been presented in this chapter. First, the complexity of problem $F_m \mid synmv, re-LU \mid C_{\max}$ is shown to be strongly NP-hard even with only one machine. For a special case, a polynomial-time algorithm which integrates the Gilmore-Gomory algorithm is proposed to solve the problem with constant loading time or unloading time ($F_1 \mid synmv, re-LU, l_j \equiv c$ or $u_j \equiv c \mid C_{\max}$). Second, a dynamic programming algorithm is formulated to effectively solve the problem in a small or medium scale. Third, two two-phase heuristic algorithms are proposed to obtain an optimal or near-optimal makespan in a reasonable CPU time. Regarding the constructive phase, two heuristics are developed: CAS and CAI. A modified neighborhood search is suggested for the improvement phase. Moreover, in order to evaluate the performance of the algorithms, a lower bound value is derived.

The experimental results show that these two-phase heuristic algorithms not only generate the makespans, which are averagely within 1% from the optimal values or 3% from the lower bounds, but also obtain solutions in a very short time. An extension to increase the number of machines in the machining center to two will be further investigated in Chapter 5.

Chapter 5

Two-machine Flow Shop with Synchronous Material Movement

5.1 Introduction

In Chapter 4, a scheduling problem with application to a T-line machining center with one CNC machine has been explored and discussed. This scheduling problem with makespan objective has been shown to be NP-hard in the strong sense. In this chapter, an extension of the T-line machining center problem with two CNC machines is considered. In Chapter 4, Figure 4.2 shows an example of this manufacturing setting with one L/U station, two CNC machine stations and a rotary table with three pallets that simultaneously moves jobs between stations in a synchronized manner. In this machining center, a job is loaded at the L/U station, processed by these two CNC machines sequentially, and finally unloaded from the machining center at the L/U station. Similar to the problem with one CNC machine, the scheduling problem of minimizing the makespan with two CNC machines can be presented in a three-field notation as $F_2 \mid synmv, re-LU \mid C_{\max}$.

The problem that is analyzed in this chapter consists of *n* jobs that have to be processed by the machining center. The loading and unloading times for job *j* are denoted as l_j and u_j, and the processing times on the first CNC machine (CNC_1) and on the second CNC machine (CNC_2) are denoted as p_{j1} and p_{j2}, respectively. A time period between two successive rotations of the rotary table is defined as a cycle time denoted as C_i where $i = 1, \ldots, n+3$. Given a job sequence

$J_{[1]}$, $J_{[2]}$, …, $J_{[n]}$, each cycle time can be represented as follows where the notation $J_{[i]}$ represents the job is sequenced in position i:

- $C_1 = l_{[1]}$; $C_2 = \max\{p_{[1]1}, l_{[2]}\}$; $C_3 = \max\{p_{[2]1}, p_{[1]2}, l_{[3]}\}$, (5.1)
- $C_i = \max\{p_{[i-1]1}, p_{[i-2]2}, l_{[i]} + u_{[i-3]}\}$, $i = 4,\dots, n$, (5.2)
- $C_{n+1} = \max\{p_{[n]1}, p_{[n-1]2}, u_{[n-2]}\}$; $C_{n+2} = \max\{p_{[n]2}, u_{[n-1]}\}$; $C_{n+3} = u_{[n]}$. (5.3)

where $l_{[i]}$, $p_{[i]1}$, $p_{[i]2}$ and $u_{[i]}$ represent the loading time, processing time on CNC$_1$, processing time on CNC$_2$ and unloading time of the job sequenced in position i, respectively.

In the first three cycles, there are no unloading operations required. Similarly, there are no loading operations performed in the last three cycles. The time duration of each cycle is equal to the maximum time required among these three stations. Figure 5.1 illustrates a schedule of jobs at each station. The problem is to determine a sequence which minimizes the summation of these cycles ($\sum_{i=1}^{n+3} C_i$).

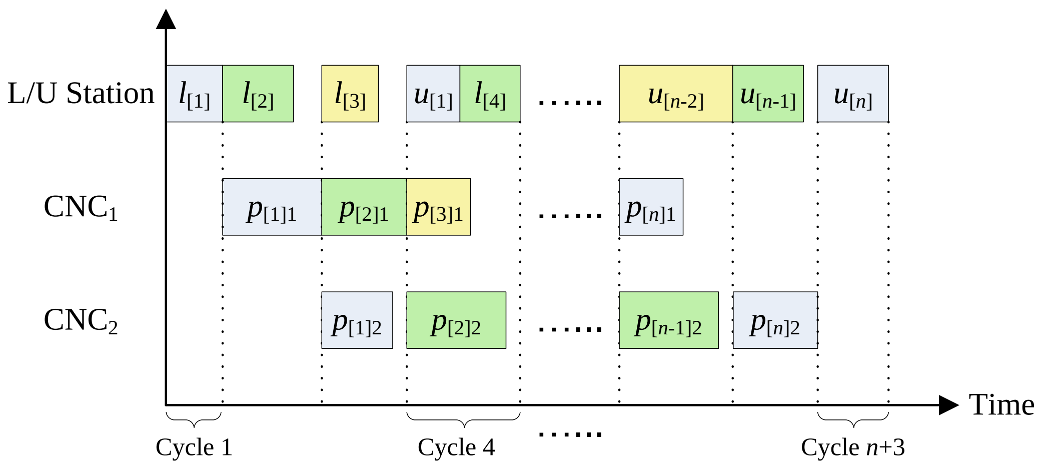


Figure 5.1. Schedule of jobs at each station in a two-CNC T-line machining center

In Section 5.2, the dynamic programming algorithm proposed in Chapter 4 is extended to this problem and a computational analysis for the algorithm is presented. Section 5.3 proposes two constructive heuristics combined with the modified neighborhood search to solve the problem in a large scale. The experimental designs and results for evaluating the heuristic algorithms are illustrated in Section 5.4. Finally, a generalized dynamic programming formulation and its computational analysis are presented for the problem with *m* machines as well as a lower bound for the problem.

5.2 Dynamic Programming Algorithm for $F_2 \mid synmv, re-LU \mid C_{\max}$

A forward dynamic programming procedure is formulated to find the minimum makespan. Given *n* jobs which have to be processed by the machining center and these jobs are numbered from 1 to *n*. Let $N = \{1, 2, \ldots, n\}$ be a set of jobs and let *S* be a subset of *N* containing the jobs that have already been processed in the machining center. Let *g* and *h* represent the jobs concurrently being processed on the CNC_2 and CNC_1 respectively, and *j* be the job being loaded at the L/U station. Then the dynamic programming formulation is as follows:

DP Algorithm for $F_2 \mid synmv, re-LU \mid C_{\max}$

- Optimal value function (OVF): $f_i(S, g, h, j)$ = minimum completion time for processing job *g* on CNC_2, processing job *h* on CNC_1, unloading the last job in *S* and loading job *j* at the L/U station, given that the *i* jobs in *S* have already been completed. (5.4)
- Optimal policy function (OPF): $p_i(S, g, h, j)$ = last job unloaded at the L/U station. Equivalently, this is also the last job added to set *S*. (5.5)

- Recurrence relation (RR):

$$f_i(S, g, h, j) = \min_{k \in S}\{ f_{i-1}(S \setminus \{k\}, k, g, h) + \max\{p_{h1}, p_{g2}, u_k + l_j\}\};\ i = 1, ..., n-3;$$

$$\{g, h, j\} \subseteq N;\ S \subseteq N \setminus \{g, h, j\},\ |S| = i. \tag{5.6}$$

- Boundary condition (BC):

$$f_0(\varnothing, g, h, j) = l_g + \max\{p_{g1}, l_h\} + \max\{p_{h1}, p_{g2}, l_j\};\ \{g, h, j\} \subseteq N. \tag{5.7}$$

- Answer (ANS): $\min_{\{g,h\} \subseteq N}\{ f_{n-2}(S, g, h, \varnothing)\}$ (5.8)

where $f_{n-2}(S, g, h, \varnothing) = \min_{k \in S}\{ f_{n-3}(S \setminus \{k\}, k, g, h) + \max\{p_{h1}, p_{g2}, u_k\}\} + \max\{p_{h2}, u_g\} + u_h;$

$$\{g, h\} \subseteq N;\ S = N \setminus \{g, h\},\ |S| = n-2. \tag{5.9}$$

Computational Effort Analysis

The computational effort of the dynamic programming algorithm is evaluated by the number of operations performed as "Addition" and "Comparison." The number of operations required for each stage of the algorithm is summarized as shown in Table 5.1.

Table 5.1. Number of operations required for each stage

Stage		Number of combinations	Addition	Comparison
Boundary condition ($i = 0$)		$n(n-1)(n-2)$	2	3
Recurrence relation ($1 \le i \le n-3$)		$n(n-1)(n-2)C_i^{n-3}$	$2i$	$2i + (i-1)$
Answer	f_{n-2} ($i = n-2$)	$n(n-1)$	n	$2(n-2)+(n-3)+1$
	Minimum makespan	1	0	$n(n-1)-1$

In the boundary condition, there are $n(n-1)(n-2)$ combinations for job g, h, and j, and each combination requires two additions and three comparisons to obtain the value for f_0. In the recurrence relation, for each i, there are $n(n-1)(n-2)$ choices for job g, h, and j, and C_i^{n-3}

combinations of jobs in set *S*. Each combination of (*S*, *g*, *h*, *j*) has *i* candidates in set *S* for *k*, and to obtain the minimum value among these *i* candidates requires extra *i* – 1 comparisons. In the answer formulation, there are *n*(*n* – 1) different (*g*, *h*) pairs for f_{n-2}, and each pair has *n* – 2 candidates for *k*. Moreover, among these *n* – 2 candidates *n* – 3 comparisons are performed to acquire the minimum value for each f_{n-2}. To obtain the minimum makespan among these f_{n-2} needs *n*(*n* – 1) –1 comparisons. Therefore, the total number of additions required is:

$$= 2n(n-1)(n-2)(1+\sum_{i=1}^{n-3} i\cdot C_i^{n-3}) + n(n-1)(n)$$

$$= 2n(n-1)(n-2)(1+(n-3)\sum_{j=0}^{n-4} C_j^{n-4}) + n^2(n-1)$$

$$= 2n(n-1)(n-2)(n-3)2^{n-4} + n(n-1)(3n-4) \approx n(n-1)(n-2)(n-3)2^{n-3}.$$

The total number of comparisons required is:

$$= n(n-1)(n-2)(3\sum_{i=1}^{n-3} i\cdot C_i^{n-3} - \sum_{i=1}^{n-3} C_i^{n-3}) + 6n(n-1)(n-2) + n(n-1) - 1$$

$$= n(n-1)(n-2)(3(n-3)2^{n-4} - 2^{n-3} + 1) + 6n(n-1)(n-2) + n(n-1) - 1$$

$$= 3n(n-1)(n-2)(n-3)2^{n-4} + n(n-1)(n-2)(7-2^{n-3}) + n(n-1) - 1$$

$$\approx 3n(n-1)(n-2)(n-3)2^{n-4}.$$

Thus, the computational effort for this dynamic programming algorithm is $O(n^4 2^{n-3})$. Consider an example for this machining center with 15 jobs. The total number of operations required for the algorithm is approximately 335 million. Furthermore, to compute a value of the optimal value function f_i, it is necessary to know and store several values of the function f_{i-1} in the previous stage. For example, in order to calculate the values of f_{10} in stage 10, we need to compute 600,600 values of f_9 in the prior stage. Therefore, the storage spaces for the algorithm would become a practical restriction for solving the problem in a large scale. Hence, it is more feasible to develop a heuristic algorithm to solve the problem in a large size.

A Numerical Example

A numerical example is presented to illustrate the proposed dynamic programming algorithm. In this example, five jobs need to be processed in a T-line machining center with two CNC machines, and one L/U station. Table 5.2 shows the loading, processing, and unloading times for these jobs.

Table 5.2. Job data for a two-CNC T-line machining center

Job	l_j	p_{j1}	p_{j2}	u_j
1	4	7	5	4
2	3	9	8	3
3	5	3	10	3
4	2	5	6	1
5	2	7	4	5

Since the process used to calculate all the values of the optimal value function is similar, only the calculations directly related to obtaining the optimal solution are shown below.

- Boundary conditions:

$$f_0(\varnothing,2,3,5) = l_2 + \max\{p_{21}, l_3\} + \max\{p_{31}, p_{22}, l_5\} = 3 + \max\{9,5\} + \max\{3,8,2\} = 20.$$

$$f_0(\varnothing,3,2,5) = l_3 + \max\{p_{31}, l_2\} + \max\{p_{21}, p_{32}, l_5\} = 5 + \max\{3,3\} + \max\{9,10,3\} = 18.$$

$$f_0(\varnothing,2,5,3) = l_2 + \max\{p_{21}, l_5\} + \max\{p_{51}, p_{22}, l_3\} = 3 + \max\{9,2\} + \max\{7,8,5\} = 20.$$

$$f_0(\varnothing,5,2,3) = l_5 + \max\{p_{51}, l_2\} + \max\{p_{21}, p_{52}, l_3\} = 2 + \max\{7,3\} + \max\{9,4,5\} = 18.$$

$f_0(\varnothing,3,5,2)=l_3+\max\{p_{31},l_5\}+\max\{p_{51},p_{32},l_2\}=5+\max\{3,2\}+\max\{7,10,3\}=18.$

$f_0(\varnothing,5,3,2)=l_5+\max\{p_{51},l_3\}+\max\{p_{31},p_{52},l_2\}=2+\max\{7,5\}+\max\{3,4,3\}=13.$

- $i=1$:

$f_1(2,3,5,1)=f_0(\varnothing,2,3,5)+\max\{p_{51},p_{32},u_2+l_1\}=20+\max\{7,10,3+4\}=30.$

$f_1(3,2,5,1)=f_0(\varnothing,3,2,5)+\max\{p_{51},p_{22},u_3+l_1\}=18+\max\{7,8,3+4\}=26.$

$f_1(2,5,3,1)=f_0(\varnothing,2,5,3)+\max\{p_{31},p_{52},u_2+l_1\}=20+\max\{3,4,3+4\}=27.$

$f_1(5,2,3,1)=f_0(\varnothing,5,2,3)+\max\{p_{31},p_{22},u_5+l_1\}=18+\max\{3,8,5+4\}=27.$

$f_1(3,5,2,1)=f_0(\varnothing,3,5,2)+\max\{p_{21},p_{52},u_3+l_1\}=18+\max\{9,4,3+4\}=27.$

$f_1(5,3,2,1)=f_0(\varnothing,5,3,2)+\max\{p_{21},p_{32},u_5+l_1\}=13+\max\{9,10,5+4\}=23.$

- $i=2$:

$f_2(\{2,3\},5,1,4)=\min\{f_1(2,3,5,1)+\max\{p_{11},p_{52},u_3+l_4\},\ f_1(3,2,5,1)+\max\{p_{11},p_{52},u_2+l_4\}\}$
$=\min\{30+\max\{7,4,3+2\},\ 26+\max\{7,4,3+2\}\}=33$; $p_2(\{2,3\},5,1,4)$ =2.

$f_2(\{2,5\},3,1,4)=\min\{f_1(2,5,3,1)+\max\{p_{11},p_{32},u_5+l_4\},\ f_1(5,2,3,1)+\max\{p_{11},p_{32},u_2+l_4\}\}$
$=\min\{27+\max\{7,10,5+2\},\ 27+\max\{7,10,3+2\}\}=37$; $p_2(\{2,5\},3,1,4)$ =2 or 5.

$f_2(\{3,5\},2,1,4)=\min\{f_1(3,5,2,1)+\max\{p_{11},p_{22},u_5+l_4\},\ f_1(5,3,2,1)+\max\{p_{11},p_{22},u_3+l_4\}\}$
$=\min\{27+\max\{7,8,5+2\},\ 23+\max\{7,8,3+2\}\}=31$; $p_2(\{3,5\},2,1,4)$ =3.

- Answer:

$\min\{f_3(\{3,4,5\},1,2,\varnothing),\ f_3(\{2,4,5\},1,3,\varnothing),\ldots,\ f_3(\{1,2,3\},5,4,\varnothing)\}=\min\{49,49,\ldots,45\}=43$

where $f_3(\{2,3,5\},1,4,\varnothing)=\min\{f_2(\{2,3\},5,1,4)+\max\{p_{41},p_{12},u_5\},$
$f_2(\{2,5\},3,1,4)+\max\{p_{41},p_{12},u_3\},$
$f_2(\{3,5\},2,1,4)+\max\{p_{41},p_{12},u_2\}\}+\max\{p_{42},u_1\}+u_4$
$=\min\{33+\max\{5,5,5\},37+\max\{5,5,3\},31+\max\{5,5,3\}\}+\max\{6,4\}+1=43$;

$p_3(\{2,3,5\},1,4,\varnothing)=2.$

Hence, the minimum makespan of the problem is 43 and the optimal job sequence is $5-3-2-1-4$.

5.3 Heuristic Algorithm for $F_2 \mid synmv, re-LU \mid C_{\max}$

Since the three-station flow shop scheduling problem is NP-hard, it is unlikely that a polynomial-time algorithm can be developed to find an optimal solution. The computational results in Chapter 4 show that the proposed two-phase heuristic algorithm performs very well for the scheduling problem with one CNC machine. As a result, the two-phase algorithm is extended to solve the problem with two CNC machines.

The proposed algorithm also consists of two stages: the constructive stage and the improvement stage. In the constructive stage, two constructive heuristics are developed. One forms an initial sequence by applying the Gilmore-Gomory algorithm (Gilmore and Gomory 1964) to the problem while neglecting the loading and unloading times. The other forms an initial sequence by inserting a job in the position of a given sequence that yields the minimum makespan. In the improvement stage, a similar algorithm to the modified neighborhood search algorithm that is proposed in Chapter 4 is employed. Furthermore, a formulation to derive a lower bound value is also presented.

Constructive Algorithm – Gilmore-Gomory Algorithm (CAGG)

When the loading and unloading times are dominated by the processing times, we only have to consider the processing times on CNC_1 and CNC_2. In this case, the problem can be regarded as a two-machine flow shop problem with blocking, which can be solved optimally by the Gilmore-Gomory algorithm. As a result, the sequence generated by the Gilmore-Gomory algorithm will be the initial seed for the improvement stage while neglecting the loading and unloading times. The makespan of the initial seed, including the loading and unloading times, will be calculated based on this sequence.

Furthermore, a lower bound value can be derived based on the assumption of neglecting the loading and unloading times. In Figure 5.1, if the cycle time of cycle i ($i = 2, \ldots, n+2$) is identified by the processing time of CNC$_1$ or CNC$_2$, the minimum value of the summation of these cycles can be obtained by applying the Gilmore-Gomory algorithm to the problem which only considers the processing times on CNC$_1$ or CNC$_2$. This minimum value plus the cycle times of the first cycle and the last cycle, which are equivalent to the smallest loading and unloading times, will be a lower bound to the original problem.

Lemma 5.1. The value, $\min_{i=1,\ldots,n} l_i + MS_{GG} + \min_{i=1,\ldots,n} u_i$, is a lower bound for the makespan problem in a T-line machining center with two CNC machines, where MS_{GG} is the optimal makespan obtained by the Gilmore-Gomory algorithm while neglecting loading and unloading times.

Proof: According to Equations (5.1) to (5.3), the makespan of a given sequence, say σ, is represented as $MS_\sigma = l_{[1]} + \max\{p_{[1]1}, l_{[2]}\} + \max\{p_{[2]1}, p_{[1]2}, l_{[3]}\} + \max\{p_{[3]1}, p_{[2]2}, u_{[1]} + l_{[4]}\} + \ldots + \max\{p_{[n-1]1}, p_{[n-2]2}, u_{[n-3]} + l_{[n]}\} + \max\{p_{[n]1}, p_{[n-1]2}, u_{[n-2]}\} + \max\{p_{[n]2}, u_{[n-1]}\} + u_{[n]}$. When the loading and unloading times are neglected from cycles 2 to n+2, cycle times of these cycles only considering processing times are the lower bounds to original cycle times (e.g., $\max\{p_{[1]1},\ l_{[2]}\} \geq p_{[1]1}$, $\max\{p_{[2]1}, p_{[1]2}, l_{[3]}\} \geq \max\{p_{[2]1}, p_{[1]2}\}$ and so on). Thus,

$$MS_{\sigma} \geq l_{[1]} + p_{[1]1} + \max\{p_{[2]1}, p_{[1]2}\} + \ldots \max\{p_{[n]1}, p_{[n-1]2}\} + p_{[n]2} + u_{[n]}$$

$$= l_{[1]} + p_{[1]1} + \sum_{i=1}^{n-1} \max\{p_{[i+1]1}, p_{[i]2}\} + p_{[n]2} + u_{[n]}.$$

In addition, $\min_{i=1,\ldots,n} l_i$ and $\min_{i=1,\ldots,n} u_i$ are the lower bounds for $l_{[1]}$ and $u_{[n]}$, respectively, and MS_{GG} is the lower bound to $p_{[1]1} + \sum_{i=1}^{n-1} \max\{p_{[i+1]1}, p_{[i]2}\} + p_{[n]2}$. Therefore,

$$MS_{\sigma} \geq \min_{i=1,\ldots,n} l_i + MS_{GG} + \min_{i=1,\ldots,n} u_i . \qquad \square$$

When the loading and unloading times on the L/U station are dominated by the processing times on CNC machines, the makespan yielded by the Gilmore-Gomory algorithm will approximate to the optimal makespan. As a result, apply the Gilmore-Gomory algorithm to generate the initial seed for the improvement stage is one of constructive heuristics proposed in this research. The heuristic algorithm is named CAGG and its procedure is described below.

CAGG Algorithm

Step 1. Apply the Gilmore-Gomory algorithm to the problem which only considers p_{j1} and p_{j2}. Assume the optimal sequence obtained by the Gilmore-Gomory algorithm is σ_G.

Step 2. Let σ_G be the initial sequence for the improvement algorithm. Calculate the makespan of sequence σ_G including the loading and unloading times.

Constructive Algorithm – Greedy Insertion (CAGI)

The computational results in Chapter 4 show that the insertion heuristic (CAI) yields a better makespan value than the selection heuristic (CAS). The makespan of the initial sequence constructed by CAI is less than 5.8% from the lower bound. Therefore, the insertion heuristic is also adopted as a constructive algorithm for the two-machine case. The insertion heuristic adopted in this chapter is called Greedy Insertion (CAGI) because only the combination of a job and an inserted position which yields the minimum makespan will be selected. When one job has to be added to the current sequence, every unscheduled job will be inserted in every position of the current sequence and the combination of the job and the position with the minimum makespan is chosen.

For example, there are 10 jobs in a problem and 4 jobs have formed a sequence. When one job is added to the current sequence, each unscheduled job (6 jobs) has to be inserted in every position of the current sequence (there are 5 positions). Thus, each unscheduled job will have five makespans corresponding to these positions. Thirty makespans will be generated and the one with the minimum value will be chosen. Therefore, given n is the number of jobs, the computational effort of the CAGI is $O(n^3)$ which can be calculated as follows:

$$\sum_{j=1}^{n} j(n-j+1) = (n+1)\sum_{j=1}^{n} j - \sum_{j=1}^{n} j^2 = \frac{n(n+1)(n+2)}{6}.$$

CAGI Algorithm

N is a set containing all jobs; *N* = {1, 2, …, *n*}. *S* is a set containing jobs which have been sequenced, and *R* is a set containing jobs which have not been sequenced. *MS* is a variable to record the current makespan.

Step 1. Let *R* = *N* and *S* = ∅.

Step 2. Select the job with the minimum makespan which is $k \in \arg \min_{j=1,\ldots,n} \{l_j + p_{j1} + p_{j2} + u_j\}$.

Let *S* = {*k*} and *R* = *N*\{*k*}.

Step 3. For every job *j* in *R*, insert it in front of job *k* and behind job *k* to obtain two makespans MS_{fj} and MS_{bj}, respectively.

MS = $\min_{j \in R}\{MS_{fj}, MS_{bj}\}$, assume the inserted job with the minimum makespan is *g* and *MS* is equal to MS_{fg}. Then job *g* is sequenced before job *k* and *S* = {*g*, *k*}. Otherwise, job *g* is sequenced after job *k* and *S* = {*k*, *g*}. *R*=*R*\{*k*}.

Step 4. For every job *j* in *R*, insert it in every position in the current sequence to obtain |*S*|+1 makespans. The makespan of inserting job *j* in position *i* in the current sequence is represented as MS_{ij} where $i = 1, \ldots, |S| + 1$.

Step 4.1.
$$\begin{aligned} MS_{ij} = MS &+ \max\{p_{[i-1]1}, p_{[i-2]2}, u_{[i-3]} + l_j\} + \max\{p_{j2}, p_{[i-1]2}, u_{[i-2]} + l_{[i]}\} \\ &+ \max\{p_{[i]1}, p_{j2}, u_{[i-1]} + l_{[i+1]}\} + \max\{p_{[i+1]1}, p_{[i]2}, u_j + l_{[i+2]}\} \\ &- \max\{p_{[i-1]1}, p_{[i-2]2}, u_{[i-3]} + l_{[i]}\} - \max\{p_{[i]1}, p_{[i-1]2}, u_{[i-2]} + l_{[i+1]}\} \\ &- \max\{p_{[i+1]1}, p_{[i]2}, u_{[i-1]} + l_{[i+2]}\} - \max\{p_{[i+2]1}, p_{[i+1]2}, u_{[i]} + l_{[i+3]}\} \end{aligned}$$

where $l_{[|S|+1]} = l_{[|S|+2]} = l_{[|S|+3]} = p_{[0]1} = p_{[|S|+1]1} = p_{[|S|+2]1} = p_{[-1]2} = p_{[0]2}$

$= p_{[|S|+1]2} = u_{[-2]} = u_{[-1]} = u_{[0]} = 0$.

Step 4.2. Select job j in position i with the minimum makespan, and insert job j in that position. $MS = \min_{j \in R, i=1,\ldots,|S|+1} \{MS_{ij}\}$ and let $S = S \cup \{j\}$, and $R = R \setminus \{j\}$. If $R \neq \varnothing$, go to step 4. Otherwise, go to step 5.

Step 5. Output the job sequence and the makespan.

A Numerical Example

A numerical example is presented to demonstrate the procedure of CAGI. Assume there are five jobs. Table 5.3 shows the job data for the example.

Table 5.3: Job data for the example of CAGI

Job	l_j	p_{j1}	p_{j2}	u_j
1	4	7	5	4
2	3	9	8	3
3	5	3	10	3
4	2	5	6	1
5	2	7	4	5

Iteration 1:

Step 1. R = {1, 2, 3, 4, 5}.

Step 2 Job 4 is selected MS = 14. S = {4}.

Iteration 2:

Step 3. R = {1, 2, 3, 5}.

Job 1: MS = min{23, 23} = 23.

Job 2: MS = min{27, 27} = 27.

Job 3: MS = min{25, 26} = 25.

Job 5: *MS* = min{21, 23} = 21.

Job 5 is selected and is inserted in front of job 4. *MS* = 21 and *S* = {5, 4}.

Iteration 3:

Step 4. *R* = {1, 2, 3}.

Step 4.1. Calculate the makespan for all jobs in *R* in every position of current sequence.

Job 1: *MS* = min{30, 28, 30} = 28.

Job 2: *MS* = min{32, 33, 34} = 32.

Job 3: *MS* = min{30, 30, 33} = 30.

Step 4.2. Job 1 is selected and is inserted in front of job 4. *MS* = 28 and *S* = {5, 1, 4}.

Iteration 4:

Step 4. *R* = {2, 3}.

Step 4.1. Calculate the makespan for all jobs in *R* in every position of current sequence.

Job 2: *MS* = min{39, 38, 40, 44} = 38.

Job 3: *MS* = min{37, 35, 40, 45} = 35.

Step 4.2. Job 3 is selected and is inserted in front of job 1. *MS* = 28 and *S* = {5, 3, 1, 4}.

Iteration 5:

Step 4. *R* = {2}.

Step 4.1. Calculate the makespan for all jobs in *R* in every position of current sequence.

Job 2: *MS* = min{49, 49, 43, 47, 49} = 43.

Step 4.2. Job 2 is selected and is inserted in front of job 1. *MS* = 28 and *S* = {5, 3, 2, 1, 4}.

Step 5. The job sequence is $5-3-2-1-4$ and the makespan is 43. (It is also optimal.)

Lower Bound for $F_2 \mid synmv, re-LU \mid C_{\max}$

Similar to Lemma 5.1, if the cycle times from cycles 2 to *n*+2 are determined by the loading and unloading times, the summation of the loading and unloading times will provide a lower bound to an optimal makespan. This property is shown in the following lemma.

Lemma 5.2. The value, $\sum_{j=1}^{n}(l_j + u_j)$, is a lower bound for the makespan problem in a T-line machining center with two CNC machines.

Proof: According to Equations (5.1) to (5.3), the makespan of a given sequence, say σ, is represented as $MS_\sigma = l_{[1]} + \max\{p_{[1]1},\ l_{[2]}\} + \max\{p_{[2]1}, p_{[1]2}, l_{[3]}\} + \max\{p_{[3]1}, p_{[2]2}, u_{[1]} + l_{[4]}\} + \ldots +$ $\max\{p_{[n-1]1}, p_{[n-2]2}, u_{[n-3]} + l_{[n]}\} + \max\{p_{[n]1},\ p_{[n-1]2}, u_{[n-2]}\} + \max\{p_{[n]2},\ u_{[n-1]}\} + u_{[n]}$. Neglect the processing times in cycles 2 to *n*+2. Thus,

$$MS_\sigma \geq l_{[1]} + l_{[2]} + l_{[3]} + (l_{[4]} + u_{[1]}) \ldots + (l_{[n]} + u_{[n-3]}) + u_{[n-2]} + u_{[n-1]} + u_{[n]}$$

$$= \sum_{j=1}^{n}(l_j + u_j).$$ □

In addition, a new lower bound which provides a tighter bound for any sequence can be derived from Lemma 5.1 and Lemma 5.2.

Theorem 5.1. The value, $\max\{\min_{i=1,\ldots,n} l_i + MS_{GG} + \min_{i=1,\ldots,n} u_i,\ \sum_{j=1}^{n}(l_j + u_j)\}$, is a lower bound for the makespan problem in a T-line machining center with two CNC machines.

Proof: Derived from Lemmas 5.1 and 5.2 directly. □

Improvement Stage

According to the computational results in Section 4.6, given an initial seed, the modified neighborhood search algorithm improves a makespan value significantly to within 3% from its corresponding lower bound. Moreover, the modified neighborhood search algorithm can generate a sequence for a problem in one second even when the number of jobs is 40. Therefore, the modified neighborhood search algorithm proposed in Chapter 4 is also adopted in this chapter. Only Step 2.1 and Step 4.1 in the procedure of the modified neighborhood search algorithm in Section 4.5 should be modified due to the two CNC machines in the problem. Step 2.1 regards the procedure of computing the new makespan after performing the adjacent pairwise interchange. Likewise, the formulation of obtaining the new makespan after performing the pairwise interchange is shown in Step 4.1. The revised procedures of these two steps are illustrated as below.

Step 2.1. Generate *S'* by swapping the positions of job *i* and job *i*+1 in *S*. The makespan can be obtained as the following formulation.

$$\mathrm{MS}(S') = \mathrm{MS}(S) - \sum_{j=i}^{i+4} \max\{p_{[j-1]1}, p_{[j-2]2}, u_{[j-3]} + l_{[j]}\} + \max\{p_{[i-1]1}, p_{[i-2]2}, u_{[i-3]} + l_{[i+1]}\}$$

$$+ \max\{p_{[i+1]1}, p_{[i-1]2}, u_{[i-2]} + l_{[i]}\} + \max\{p_{[i]1}, p_{[i+1]2}, u_{[i-1]} + l_{[i+2]}\}$$

$$+ \max\{p_{[i+2]1}, p_{[i]2}, u_{[i+1]} + l_{[i+3]}\} + \max\{p_{[i+3]1}, p_{[i+2]2}, u_{[i]} + l_{[i+4]}\}$$

where $l_{[n+1]} = l_{[n+2]} = l_{[n+3]} = p_{[0]1} = p_{[n+1]1} = p_{[n+2]1} = p_{[-1]2} = p_{[0]2}$

$= p_{[n+1]2} = u_{[-2]} = u_{[-1]} = u_{[0]} = 0$.

Step 4.1. Generate *S'* by swapping the job positions of job *i* and job *j* where $j \geq i+2$ in *S*. The makespan can be obtained as the following formulation.

$$\mathrm{MS}(S') = \mathrm{MS}(S) - \sum_{k=i}^{i+3} \max\{p_{[k-1]1}, p_{[k-2]2}, u_{[k-3]} + l_{[k]}\} - Q$$

$$+\max\{p_{[i-1]1}, p_{[i-2]2}, u_{[i-3]} + l_{[j]}\} + \max\{p_{[j]1}, p_{[i-1]2}, u_{[i-2]} + l_{[i+1]}\}$$

$$+\max\{p_{[j+1]1}, p_{[i]2}, u_{[j-1]} + l_{[j+2]}\} + \max\{p_{[j+2]1}, p_{[j+1]2}, u_{[i]} + l_{[j+3]}\} + V$$

where $l_{[n+1]} = l_{[n+2]} = l_{[n+3]} = p_{[0]1} = p_{[n+1]1} = p_{[n+2]1} = p_{[-1]2} = p_{[0]2}$

$= p_{[n+1]2} = u_{[-2]} = u_{[-1]} = u_{[0]} = 0$, and Q and V are computed as follows.

If $j = i+2$

$$Q = \sum_{k=j+2}^{j+3} \max\{p_{[k-1]1}, p_{[k-2]2}, u_{[k-3]} + l_{[k]}\}.$$

$$V = \max\{p_{[i+1]1}, p_{[j]2}, u_{[i-1]} + l_{[i]}\} + \max\{p_{[i]1}, p_{[i+1]2}, u_{[j]} + l_{[i+3]}\}.$$

Else If ($j = i+3$)

$$Q = \sum_{k=j+1}^{j+3} \max\{p_{[k-1]1}, p_{[k-2]2}, u_{[k-3]} + l_{[k]}\}.$$

$$V = \max\{p_{[i+1]1}, p_{[j]2}, u_{[i-1]} + l_{[i+2]}\} + \max\{p_{[i+2]1}, p_{[i+1]2}, u_{[j]} + l_{[i]}\}$$

$$+\max\{p_{[i]1}, p_{[j-1]2}, u_{[j-2]} + l_{[j+1]}\}.$$

Else

$$Q = \sum_{k=j}^{j+3} \max\{p_{[k-1]1}, p_{[k-2]2}, u_{[k-3]} + l_{[k]}\}.$$

$$V = \max\{p_{[i+1]1}, p_{[j]2}, u_{[i-1]} + l_{[i+2]}\} + \max\{p_{[i+2]1}, p_{[i+1]2}, u_{[j]} + l_{[i+3]}\}$$

$$+\max\{p_{[j-1]1}, p_{[j-2]2}, u_{[j-3]} + l_{[i]}\} + \max\{p_{[i]1}, p_{[j-1]2}, u_{[j-2]} + l_{[j+1]}\}.$$

5.4 Computational Results

In order to evaluate the performance of the proposed heuristic algorithms, a series of the experiments is conducted. The solutions generated by the two developed algorithms are compared with the optimal solutions obtained by the dynamic programming algorithm in Section 5.2. All of the data for the experiments are randomly generated in this research. Similar to the experiments in Chapter 4, three different scenarios are examined with respect to the loading, processing and unloading times. In addition, for each scenario, three sizes of the problems are considered where small-size problems with 10-job and 15-job are examined. The makespan values and CPU times are recorded for all experiments. The experimental design and the rules to generate the testing data are summarized in Table 5.4.

Table 5.4: Experimental design and data generating rules for $F_2 \mid synmv, re-LU \mid C_{\max}$

	Small size	Medium size	Large size
Number of jobs (*n*)	10/15	17	40
Scenario I (7, 11, 11, 3)	$(l_j, p_{j1}, p_{j2}, u_j)$ = (U(1,7), U(1, 11), U(1, 11), U(1, 3))		
Scenario II (7, 15, 15, 3)	$(l_j, p_{j1}, p_{j2}, u_j)$ = (U(1,7), U(1, 15), U(1, 15), U(1, 3))		
Scenario III (10, 11, 11, 4)	$(l_j, p_{j1}, p_{j2}, u_j)$ = (U(1,10), U(1, 11), U(1, 11), U(1, 4))		

*U denotes the discrete distribution and all operation times are integer.

Two constructive algorithms, CAGG and CAGI, are developed and each of them is integrated with the modified neighborhood search algorithm as one two-phase algorithm. Thus, two two-phase algorithms are implemented and they are named as CAGG_M and CAGI_M, respectively. These two heuristic algorithms are implemented in Borland C++ 5.5 as well as the dynamic programming algorithm. Ten runs are executed for each scenario and these tests are run on a Pentium 1.40GHz PC with 1 GB RAM. The maximum number of jobs in the problem can be solved optimally by the dynamic programming algorithm is 17 even on an Intel Core 2 Duo 1.6GHz PC with 3 GB RAM. Hence, the number of jobs in the medium-size problem is set to 17.

Results

Similar to the experiments conducted in Chapter 4, there are three scenarios with the four different numbers of jobs in problems. Ten runs are executed for each case. Hence, 120 instances are tested for these two proposed heuristics. For small and medium size problems, optimal solutions can be obtained by the dynamic programming algorithm presented in Section 5.2. In a large-size problem, however, only lower bound values derived from Theorem 5.1 will be bases to compare with makespans obtained by the heuristic algorithms. The average makespan on 10 runs for each case is summarized in Table 5.5. Table 5.6 illustrates the average relative errors from the optimums and lower bounds.

Table 5.5: Summary of the average makespans obtained by the DP, heuristics, and LB

Scenario	*n*	DP		CAGG_M			CAGI_M			LB
		Optimum	Time	S1	S2	Time	S1	S2	Time	
I (7, 11, 11, 3)	10	81.2	0.24	89.3	82	0.16	83	82.2	0.16	77.5
	15	105.5	30.46	118.1	107.5	0.34	110.2	107.3	0.31	101.4
	17	126.5	201.2	143.6	129.4	0.42	130.7	129.2	0.42	122.2
	40	–	–	309.2	271.1	1.98	279.8	270.5	1.98	254.7
II (7, 15, 15, 3)	10	99.8	0.20	106.4	101.4	0.15	101.8	100.9	0.17	95.7
	15	142.3	30.46	154.4	145.4	0.30	147.3	144.0	0.28	138.2
	17	161.7	200.6	170.9	164.2	0.39	166.4	163.2	0.33	158.2
	40	–	–	378.9	355	1.89	361.5	354.2	1.94	344.6
III (10, 11, 11, 4)	10	84.8	0.19	95.3	86.6	0.20	86.1	85.4	0.15	81.7
	15	125.4	30.54	144	128.8	0.32	130.7	128.2	0.32	122.4
	17	142	202.0	161.9	143.6	0.40	146.4	142.9	0.38	141
	40	–	–	374	325.4	1.74	331.9	324.2	1.69	321.7

(S1: the constructive stage; S2: the improvement stage)

Table 5.6: Summary of average relative errors from optimum and LB

Scenario	*n*	RE from optimum (%)				RE from LB (%)			
		CAGG_M		CAGI_M		CAGG_M		CAGI_M	
		S1	S2	S1	S2	S1	S2	S1	S2
I (7, 11, 11, 3)	10	10.14	1.05	2.27	1.25	15.49	5.93	7.24	6.16
	15	11.98	1.92	4.45	1.73	16.60	6.11	8.76	5.91
	17	13.69	2.33	3.37	2.18	17.79	6.00	7.08	5.86
	40	–	–	–	–	21.50	6.47	9.93	6.25
II (7, 15, 15, 3)	10	6.66	1.63	2.07	1.15	11.30	6.04	6.50	5.53
	15	8.61	2.22	3.60	1.24	11.92	5.34	6.77	4.32
	17	5.89	1.64	3.00	0.97	8.41	4.04	5.46	3.36
	40	–	–	–	–	10.06	3.07	4.93	2.82
III (10, 11, 11, 4)	10	12.41	2.22	1.56	0.72	16.89	6.30	5.60	4.71
	15	14.95	2.73	4.26	2.26	17.96	5.39	6.98	4.91
	17	14.11	1.23	3.17	0.70	15.08	2.09	4.04	1.54
	40	–	–	–	–	16.40	1.21	3.25	0.82

In the small-size problems, the optimal solutions can be obtained by the dynamic programming algorithm within 0.24 and 30.5 seconds for 10-job and 15-job cases, respectively. The execution time of these two heuristic algorithms is less than 0.34 seconds for all runs. In 10-job cases, for algorithm CAGG_M, the average relative error from the optimal makespan is less than 1.6%; for algorithm CAGI_M, this value is less than 1%. When the number of jobs increases to 15, the average relative errors from optimal makespans increase to 2.3% and 1.7%, respectively. The relative errors from optimal values of the initial seed constructed in the constructive stage by CAGG are averagely better than the values obtained by CACI by five times.

When the number of jobs increases to 17, it requires around 200 seconds on an Intel Core 2 Duo 1.6GHz PC with 3 GB RAM for the dynamic programming algorithm to solve the problem. However, these two heuristics only require 0.4 seconds to obtain a solution, which is significantly faster than the dynamic programming algorithm. The relative error from the optimal makespan is less than 2.3% for scenario I and less than 1.6% in the cases of Scenario II and III. Moreover, based on the measurement of relative errors, CAGI_M is outperformed by CAGG_M in both constructive and improvement stages.

For large-size problems, because optimal solutions are unlikely to be obtained, the makespans generated by the heuristic algorithms are only compared with lower bound values. By observing the relative errors from the lower bounds, relative errors decrease as the number of jobs increases in the cases of Scenario II and III. The values are less than 3% when $n = 40$. For Scenario I, the relative errors from the lower bound values remain around 6% regardless of the number of jobs. In the small and medium size problems, the relative errors from the optimal values are less than 2.7%. As a result, this may imply that the relative errors from the optimal makespans when $n = 40$ are also less than 2.7%. Furthermore, the CPU times required by these two algorithms are around two seconds. Hence, the proposed algorithms can reach solutions rapidly even when the number of jobs is large.

Conclusions

The relative errors of the makespans obtained by these two proposed algorithms CAGG_M and CAGI_M compared with the optimal solutions are on average within 2.7% and compared with the lower bounds are 6.5% in the worse case. Moreover, optimal sequences can be found in most runs in the cases of Scenario II and III, especially when the number of jobs is large. With respect to the constructive stage, solutions formed by CAGI are much better than those by CAGG. The modified neighborhood search algorithm significantly improves the makespan value on the initial sequence. Regarding the computational effort, the CPU time is not a concern to solve a large-size problem by the proposed algorithms. Computational results in this section and in Section 4.6 indicate that the two-phase algorithm, which combines the insertion heuristic in the constructive phase and the modified neighborhood search algorithm in the improvement phase, is applicable to solve the minimizing makespan problem of a T-line machining center.

Additionally, the lower bound derived from Theorem 5.1 provides a good insight about the optimal makespan when the optimum is unavailable. Observing the results of Scenario II in the large-size problems, when most processing times on CNC machines dominate the summation of most pairs of loading and unloading times, a lower bound can be obtained by applying the Gilmore-Gomory algorithm. Conversely, in Scenario III, if the sum of the loading and unloading times is much greater than the summation of processing times on both of machines, then the lower bound, $\sum_{j=1}^{n}(l_j + u_j)$, will approximate to the optimal value.

5.5 Dynamic Programming Algorithm for $F_m \mid synmv, re-LU \mid C_{\max}$

In this section, the proposed dynamic programming algorithm for the scheduling problem in a T-line machining center with two machines is generalized to a T-line machining center with *m* machines. The scheduling problem of an *m*-machine T-line machining center is denoted as $F_m \mid synmv, re-LU \mid C_{\max}$. Given *n* jobs which have to be processed by the machining center and these jobs are numbered from 1 to *n*. Let $N = \{1, 2, \ldots, n\}$ be the set of jobs and let *S* be a subset of *N* containing the jobs that have already been processed in the machining center. Let Ψ be a subset of *N* containing the jobs currently being processed on these *m* machines, and *j* be the job being loaded at the L/U station. In addition, the first job in Ψ denoted as $\Psi_{(1)}$ is the job being processed on the m^{th} machine, the second job $\Psi_{(2)}$ is the job being processed on the m–1th machine, and so on. Then, the generalized dynamic programming formulation is as follows.

DP Algorithm for $F_m \mid synmv, re-LU \mid C_{\max}$

- Optimal value function (OVF): $f_i(S, \Psi, j)$ = minimum completion time for processing jobs in Ψ on machines, unloading the last job in S and loading job j at the L/U station, given that the i jobs in S have already been completed. (5.10)
- Optimal policy function (OPF): $p_i(S, \Psi, j)$ = last job unloaded at the L/U station. Equivalently, this is also the last job added to set S. (5.11)
- Recurrence relation (RR):

$$f_i(S,\Psi,j) = \min_{k\in S}\{f_{i-1}(S\setminus\{k\},\{k\}\cup\Psi\setminus\Psi_{(m)},\Psi_{(m)}) + \max\{p_{\Psi_{(m)}1}, p_{\Psi_{(m-1)}2},\ldots,p_{\Psi_{(1)}m},u_k+l_j\}\};$$

$$i=1,\ldots,n-m-1;\ \{\Psi,j\}\subseteq N;\ S\subseteq N\setminus\{\Psi,j\},\ |S|=i. \qquad (5.12)$$

- Boundary condition (BC):

$$f_0(\varnothing,\Psi,j) = \sum_{i=1}^{m}\max\{p_{\Psi_{(i-1)}1}, p_{\Psi_{(i-2)}2},\ldots,p_{\Psi_{(i-m)}m}, l_{\Psi_{(i)}}\}$$

$$+\max\{p_{\Psi_{(m)}1}, p_{\Psi_{(m-1)}2},\ldots,p_{\Psi_{(1)}m}, l_j\};\ \{\Psi,j\}\subseteq N. \qquad (5.13)$$

- Answer (ANS): $\min_{\Psi\subseteq N}\{f_{n-m}(S,\Psi,\varnothing)\}$ (5.14)

where $f_{n-m}(S,\Psi,\varnothing) = \min_{k\in S}\{f_{n-m-1}(S\setminus\{k\},\{k\}\cup\Psi\setminus\Psi_{(m)},\Psi_{(m)}) + \max\{p_{\Psi_{(m)}1},\ldots,p_{\Psi_{(1)}m},u_k\}\}$

$$+\sum_{i=1}^{m}\max\{p_{\Psi_{(i+m)}1}, p_{\Psi_{(i+m-1)}2},\ldots,p_{\Psi_{(i+1)}m}, u_{\Psi_{(i)}}\};\ \Psi\subseteq N;\ S=N\setminus\Psi,\ |S|=n-m. \qquad (5.15)$$

Computational Effort Analysis

The computational effort of this generalized dynamic programming algorithm is evaluated by the number of operations performed as “Addition” and “Comparison.” The number of operations required for each stage of the algorithm is summarized in Table 5.7.

Table 5.7. Number of operations required for each stage

Stage		Number of combinations	Addition	Comparison
BC ($i = 0$)		$(m+1)!C_{m+1}^{n}$	m	$m(m+1)/2$
RR ($1 \le i \le n-m-1$)		$(m+1)!C_{m+1}^{n}C_{i}^{n-(m+1)}$	$2i$	$m\times i + (i-1)$
ANS	$f_{n-m}(i=n-m)$	$m!C_{m}^{n}$	n	$m(n-m)+(n-m-1)+m(m-1)/2$
	Min makespan	1	0	$m!C_{m}^{n}-1$

In the boundary condition, there are $(m+1)!C_{m+1}^{n}$ combinations for *m* jobs in Ψ and job *j*. Each combination requires *m* additions and $m(m+1)/2$ comparisons to obtain the value for f_0. In the recurrence relation, for each *i*, there are also $(m+1)!C_{m+1}^{n}$ choices for *m* jobs in Ψ and job *j*, and C_{i}^{n-m-1} combinations of jobs in set *S*. Each combination of (*S*, Ψ, *j*) has *i* candidates in set *S* for *k*. Obtaining the minimum value among these *i* candidates requires extra *i* – 1 comparisons. In the answer formulation, there are $m!C_{m}^{n}$ different combinations in Ψ for f_{n-2}, and each combination has $n-m$ candidates for *k*. Moreover, among these $n-m$ candidates $n-m-1$ comparisons are performed to acquire the minimum value for each f_{n-m}. To obtain the minimum makespan among these f_{n-m} needs $m!C_{m}^{n}-1$ comparisons. Hence, the total number of additions required is:

$$= (m+1)!C_{m+1}^{n}(m+2\sum_{i=1}^{n-m-1} i\cdot C_i^{n-m-1}) + m!nC_m^n$$

$$= (m+1)!C_{m+1}^{n}(m+\frac{n}{n-m}+2(n-m-1)\sum_{j=1}^{n-m-2} C_j^{n-m-2})$$

$$= (m+1)!C_{m+1}^{n}(m+\frac{n}{n-m}+2(n-m-1)2^{n-m-2})$$

$$\approx \frac{n!}{(n-m-2)!}2^{n-m-1}.$$

The total number of comparisons required is:

$$= (m+1)!C_{m+1}^{n}(\frac{m(m+1)}{2}+(m+1)\sum_{i=1}^{n-m-1} i\cdot C_i^{n-m-1} - \sum_{i=1}^{n-m-1} C_i^{n-m-1}) + m!C_m^n\delta - 1$$

$$(\delta = (n-m)(m+1)+m(m-1)/2)$$

$$= (m+1)!C_{m+1}^{n}(\frac{m(m+1)}{2}+(m+1)(n-m-1)2^{n-m-2}-2^{n-m-1}+1) + m!C_m^n\delta - 1$$

$$= (m+1)!C_{m+1}^{n}(\frac{m(m+1)}{2}+(m+1)(n-m-1)2^{n-m-2}-2^{n-m-1}+1+\frac{\delta}{n-m}) - 1$$

$$\approx \frac{n!(m+1)}{(n-m-2)!}2^{n-m-2}.$$

Hence, the computational effort for the generalized dynamic programming algorithm is $O(\frac{n!m}{(n-m-2)!}2^{n-m-2})$.

Lower Bound for $F_m \mid synmv, re-LU \mid C_{\max}$

According to the computational results in Sections 4.6 and 5.4, the derived lower bound value provides not only a useful insight about the optimal makespan but also a value for evaluating the quality of a solution obtained by a heuristic algorithm. When the summation of loading and unloading times dominate the summation of the processing times, the value $\sum_{j=1}^{n}(l_j + u_j)$ is shown to be the lower bound for any sequence as Lemmas 4.2 and 5.2. It is also valid for the *m* machine case.

On the other hand, if the cycle time except the first and last cycle is identified by processing times on the machines, the summation of processing times in one-machine problem and the makespan obtained by the Gilmore-Gomory algorithm in the two-machine problem are the lower bounds as shown in Lemmas 4.1 and 5.1, respectively. For a general case with *m* machines, a lower bound can be obtained by relaxing the constraint of the order of a job processed by the machines. It means that a job can be processed by machines in any order.

Figure 5.2 illustrates a schedule of *n* jobs on the L/U stations and *m* machines when $n > m$. Since the cycle times for cycles 2 to *n*+*m* are determined by the processing times, these cycles are partitioned into three blocks. The first block is from cycle 2 to cycle *m*, the second block is from cycle *m*+1 to cycle *n*+1, and the third block is from cycle *n*+2 to cycle *n*+*m*. With relaxing job orders and neglecting loading and unloading times, an algorithm is proposed to minimize the total cycle time of cycles in these three blocks. The value obtained by the proposed algorithm is the optimal makespan for the relaxed problem and will be a lower bound for the original problem. This relaxed problem has been discussed by Soylu *et al.* (2007). However, their procedure to obtain the solution can be further improved by the proposed algorithm in this research.

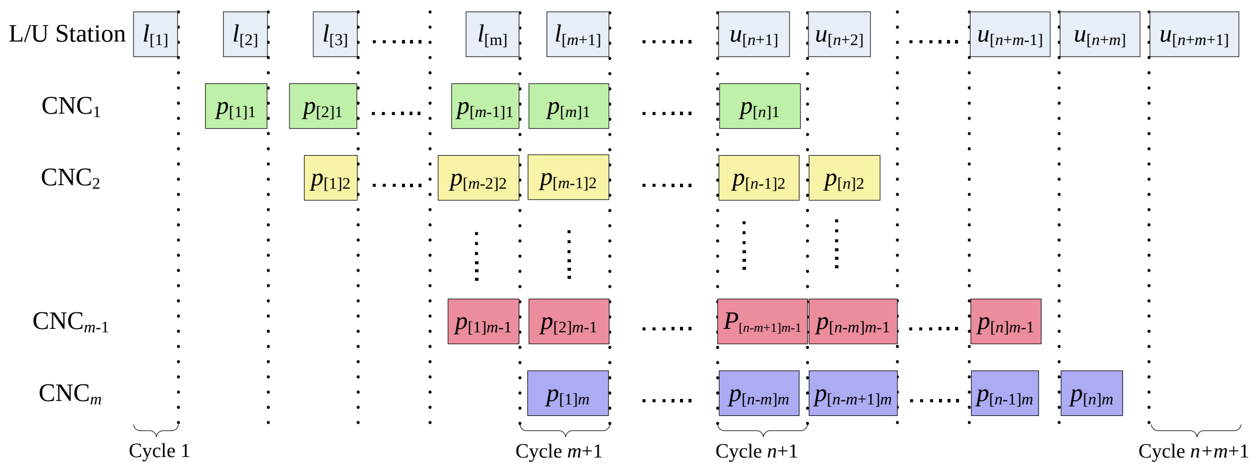


Figure 5.2. Schedule of jobs at each station in an m-CNC T-line machining center

The idea of the algorithm is to assign $n-m+1$ largest processing times on each machine to the second block (cycle m+1 to cycle n+1). Then, the rest of $m-1$ smallest processing times on each machine are assigned to corresponding cycles in the first and third blocks. The principle to assign processing times to a cycle is based on the rankings of processing times. For example, all the largest processing times on each machine are assigned to the same cycle. An algorithm is proposed to assign processing times to each cycle so that the summation of the cycle times (cycles 2 to n+m) is minimized. The procedure to obtain the optimal makespan (denoted as LB_R) for the relaxed problem is described as below.

Algorithm for LB_R

Let p_k^h be the hth largest processing time on machine k where $1 \le h \le n$ and $1 \le k \le m$. Let C_i be the time length of cycle i where $2 \le i \le n+m$. Let set R_h consist of the hth largest processing times on all machines, which is $R_h = \{p_k^h \mid 1 \le k \le m\},\ 1 \le h \le n$.

Step 1. Let $C_i = 0,\ i = 2, \ldots, n+m$.

Step 2. For h=1 to n-m+1.

Step 2.1. Select the largest processing time from set R_h, say p_k^h. p_k^h is the h^{th} largest processing time on machine k. Let $i = h + m$.

Step 2.2. Assign p_k^h to cycle i, and $C_i = p_k^h$. Also, assign the rest of processing time from set R_h to corresponding machines in cycle i.

Step 2.3. Let $i = i + 1$. Go to Step 2.1.

Step 3. For h= n-m+2 to n. Let $r = m$ and $s = n + 2$. Let $u = 1$ and $v = m$.

Step 3.1. Assign p_k^h in set R_h to machine k in cycle g_1 where $g_1 = r - u + k,\ k = 1,\ldots,u$. Let $C_{g1} = \max\{C_{g1},\ p_k^h\}$ and $R_h = R_h \setminus \{p_k^h\}$. Also, Assign p_f^h in set R_h to machine f in cycle g_2 where $g_2 = s + v - (m + 1 - f),\ f = m,\ldots,m - v + 1$. Let $C_{g2} = \max\{C_{g2},\ p_f^h\}$ and $R_h = R_h \setminus \{p_f^h\}$. If $h = n$, go to Step 4.

Step 3.2. If $C_r \geq C_s$, assign the rest of processing times in set R_h to corresponding machines in cycle r. Update the cycle time of cycle r as $C_r = \max\limits_{1 \leq k \leq m-v} \{p_k^h\}$. Let $r = r - 1$ and $v = v + 1$.

Step 3.3. If $C_r < C_s$, assign the rest of processing times in set R_h to corresponding machines in cycle s. Update the cycle time of cycle s as $C_s = \max\limits_{u+1 \leq k \leq m} \{p_k^h\}$. Let $s = s + 1$ and $u = u + 1$.

Step 3.4. Let $h = h + 1$. Go to Step 3.1.

Step 4. $LB_R = \sum_{i=2}^{n+m} C_i$.

A Numerical Example

A numerical example is presented to demonstrate the procedure of the algorithm to obtain the optimal value LB_R. Assume there are eight jobs with four machines. Table 5.8 shows the job data for the example.

Table 5.8: Job data for the example of the algorithm to obtain LB_R

Job	p_{j1}	p_{j2}	p_{j3}	p_{j4}
1	5	2	6	3
2	11	8	7	15
3	9	15	6	11
4	7	4	13	2
5	4	13	1	11
6	8	14	9	10
7	11	8	8	7
8	14	15	8	1

Initialization: without considering the job order processed by machines, processing times on each machine are sort in descending order as Table 5.9.

Table 5.9: Job data for the example after sorting in descending order

Set	p_{j1}	p_{j2}	p_{j3}	p_{j4}
R_1	14	15	13	15
R_2	11	15	9	11
R_3	11	14	8	11
R_4	9	13	8	10
R_5	8	8	7	7
R_6	7	8	6	3
R_7	5	4	6	2
R_8	4	2	1	1

Step 2: all processing times in sets R_1 to R_5 are assigned to cycles 5 to 9. Thus, $C_5 = 15$, $C_6 = 15$, $C_7 = 14$, $C_8 = 13$ and $C_9 = 8$.

Step 3: For *h*= 6 to 8.

- Iteration 1 (*h*=6): assign $p_1^6 = 7$ to cycle 4 and $p_4^6 = 3$ to cycle 10 as shown in Figure 5.3. Since $C_4 > C_{10}$, the rest of processing times in set R_6 are assigned to cycle 4 and $C_4 = 8$.

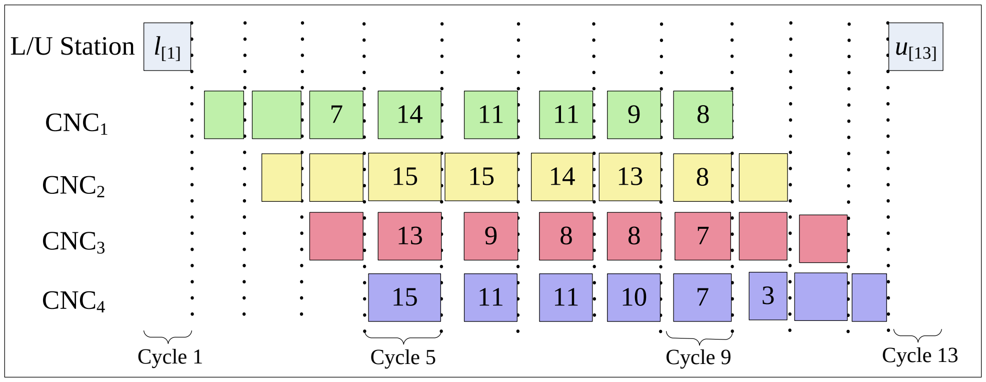


Figure 5.3. Schedule of jobs for the relaxed problem with 4 machines (*h*=6)

- Iteration 1 (*h*=7): assign $p_1^7 = 5$ to cycle 3, $p_3^7 = 6$ to cycle 10 and $p_4^7 = 2$ to cycle 11 as shown in Figure 5.4. Since $C_3 < C_{10}$, the rest of processing times in set R_7 are assigned to cycle 10 and $C_{10} = 6$.

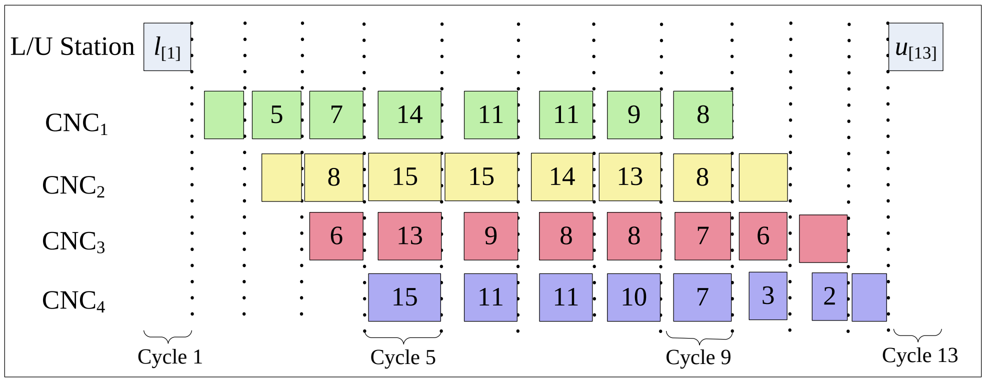


Figure 5.4. Schedule of jobs for the relaxed problem with 4 machines (*h*=7)

- Iteration 1 (h=8): assign the processing times in R_8 $p_1^8 = 4$, $p_2^8 = 2$, $p_3^8 = 1$ and $p_4^8 = 1$ to cycles 1, 2, 11 and 12, respectively as shown in Figure 5.5. Since h=n, go to Step 4.

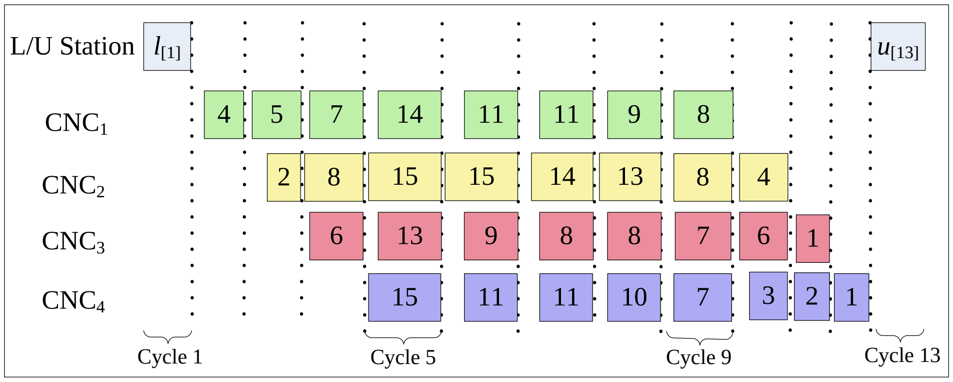


Figure 5.5. Schedule of jobs for the relaxed problem with 4 machines (h=8)

Step 4: the optimal makespan of the relaxed problem is 91. ($LB_R = \sum_{i=2}^{12} C_i = 91$)

Lemma 5.3. The value LB_R is the optimal makespan for the relaxed problem without considering jobs orders processed by machines and loading and unloading times.

Proof: For each machine, we will show that any interchange of two processing times will not decrease LB_R. The similar proof can be found in the paper by Soyleu *et al.* (2007). Assume two processing times in cycles r and s on machine k are interchanged and $C_r \le C_s$. Let these two processing times in cycles r and s be p_k^r and p_k^s, respectively. Since $C_r \le C_s$, p_k^r is not greater than p_k^s according the proposed algorithm. After the interchange, the cycle times of cycles r and s are represented as $C_r^{'}$ and $C_k^{'}$. Four cases have to be considered as follows:

- $C_r = p_k^r$, $C_s = p_k^s$. After the pairwise interchange, $C_r^{'} = p_k^s = C_s$ and $C_r = p_k^r \le C_s^{'}$. Thus, $C_r + C_s \le C_r^{'} + C_s^{'}$.

- $C_r = p_k^r,\ C_s > p_k^s$. After the pairwise interchange, $C_r \le C_r^{'} = p_k^s$ and $C_s = C_s^{'}$. Thus, $C_r + C_s \le C_r^{'} + C_s^{'}$
- $C_r > p_k^r,\ C_s = p_k^s$. After the pairwise interchange, $C_r^{'} = C_s = p_k^s$ and $C_s^{'} = \max\{p_k^r,\ \max_{f \ne k}\{p_f^s\}\}$. p_k^s is the largest processing time in cycle s and $C_r \le C_s$, which implies that the cycle time of cycle s even excluding p_k^s is still greater than or equal to C_r . Otherwise, p_k^s is assigned to cycle r rather than cycle s. Thus, $C_s^{'} = \max_{f \ne k}\{p_f^s\} \ge C_r$ so that $C_r + C_s \le C_r^{'} + C_s^{'}$
- $C_r > p_k^r,\ C_s > p_k^s$. After the pairwise interchange, $C_r^{'} = \max\{C_r,\ p_k^s\} \ge C_r$ and $C_s = C_s^{'}$. Thus, => $C_r + C_s \le C_r^{'} + C_s^{'}$

In all cases, the cycle times cannot be improved by the pairwise interchange. The makespan generated by the proposed algorithm for the relaxed problem is optimal. □

Theorem 5.2. The value, $\max\{\sum_{j=1}^{n}(l_j + u_j),\ \min_{i=1,\dots,n} l_i + LB_R + \min_{i=1,\dots,n} u_i\}$, provides a lower bound for the makespan scheduling problem of a T-line machining center with *m* CNC machines where LB_R is the optimal makespan for the relaxed problem without considering job orders processed by machines and loading and unloading times.

Proof: The value $\sum_{j=1}^{n}(l_j + u_j)$ can be shown to be a lower bound using arguments similar to those in Lemmas 4.2 and 5.2. From Lemma 5.3, LB_R is the optimal makespan for the relaxed problem which only considers processing times on machines and relaxes jobs' processing orders on machines. In addition, $\min_{i=1,\dots,n} l_i$ and $\min_{i=1,\dots,n} u_i$ are the lower bounds for $l_{[1]}$ and $u_{[n]}$, respectively.

Thus, $\min_{i=1,\ldots,n} l_i + LB_R + \min_{i=1,\ldots,n} u_i$ is a lower bound for the problem $F_m \mid synmv, re-LU \mid C_{\max}$.

Therefore, $\max\{\sum_{j=1}^{n}(l_j + u_j),\ \min_{i=1,\ldots,n} l_i + LB_R + \min_{i=1,\ldots,n} u_i\}$ is a lower bound. □

5.6 Concluding Summary

In conclusion, the scheduling problem with the makespan objective in a T-line machining center with two machines can be solve optimally by the proposed dynamic programming algorithm in $O(n^4 2^{n-3})$. Furthermore, the formulation of the dynamic programming algorithm has been extended to a general case with *m* machines and the analysis of its computational effort is also conducted. Two constructive heuristics and the modified neighborhood search are combined respectively as two two-phase algorithms to solve the scheduling problem with two machines. The computational results show that these two algorithms efficiently achieve optimal or near-optimal solutions for a wide range of test cases, especially algorithm CAGI_M. In a large-size problem ($n = 40$), CAGI_M only requires 2 seconds to obtain a solution. Additionally, the average relative error from the optimal makespan is 2.3% and from the lower bound is 6.3% for the worse case.

Another contribution is that lower bound formulations has been developed for both two-machine and *m*-machine cases. For the two-machine case according to the computational results, the suggested lower bound is within 4.6% from the optimal value in the case of Scenario I. This implies that the derived lower bound can be a reliable basis to measure the quality of a solution obtained by heuristics when the optimum is not available. For the *m*-machine case, a procedure to generate a lower bound value is also presented.

Chapter 6

Summary and Future Study

6.1 Summary

This dissertation has addressed two types of flow shop scheduling problems considering asynchronous and synchronous transportation times: (1) *n*-job, two-machine flow shop with a single transporter, and (2) *n*-job in a T-line machining center with one CNC machine. Additional extension of the T-line machining center with two CNC machines has also been discussed. The research has developed algorithms to obtain a job schedule that minimizes the makespan value for each problem.

The first type of the problem, $TF_2 \mid p_{j1} \equiv p_1, v = 1, c \geq u \mid C_{\max}$, is a special case of two-machine flow shop scheduling problem. The problem assumes the processing times for all jobs on the first machine are identical and the capacity of the transporter is greater than or equal to a threshold value derived from Property 3.2. The methodology of dynamic programming is applied to the problem. In the dynamic programming formulation, only the integer departure points are considered. The algorithm can determine the optimal schedule of transporting jobs from the first machine to the second machine in polynomial time. The complexity of the proposed dynamic programming algorithm is shown as $O(n^3)$. This is better than the complexity of the algorithm, $O(c^3n^3)$, developed by Lee and Chen (2001) given *c* is the capacity of the transporter and *n* is the number of jobs.

For the problem with synchronous transportation times $F_m \mid synmv, re-LU \mid C_{\max}$, sequencing jobs in a T-line machining center with a single machine (*m*=1) has been shown to be NP-hard in the strong sense. That is, the problem is inherently intractable and it is unlikely to develop a polynomial time algorithm to find an optimal solution. A polynomial time algorithm, $O(n^3 \log n)$, is proposed to solve a special case when the loading or unloading times are constant (denoted as $F_1 \mid synmv, re-LU, u_j \equiv c \mid C_{\max}$). The proposed algorithm incorporates the Gilmore-Gomory algorithm which can optimally solve the two-machine flow shop problem with blocking. For the general setting of the one-machine problem in a small or medium scale, a dynamic programming algorithm is developed to obtain an optimal solution efficiently. Due to the complexity of the problem, however, the dynamic programming algorithm becomes impractical to solve a large- size problem – the algorithm would consume extremely amount of storage space.

In order to solve the one-machine problem in a large scale, heuristic algorithms are developed to obtain near-optimal solutions in a reasonable CPU time. The methodology of the two-phase algorithm is adopted. Two constructive heuristics, CAS and CAI, are provided to build an initial sequence. In the improvement stage, a modified neighborhood search algorithm is developed to refine the solution obtained in the constructive stage. The pairwise interchange is applied to generate a list of neighborhood sequences. The mechanism of randomly selecting a sequence with the same makespan value as the current solution to become a new seed is incorporated in the algorithm when no improvement is gained. The purpose of the random selection is to prevent trapping the solution at a local optimum and to attempt to move the search direction to unexplored space.

In order to evaluate the performance of the heuristic algorithms, a series of experiments are conducted. These testing settings consist of different sizes of the problems (small, medium, and large) combined with three configurations on loading, processing, and unloading times. The computational results show that these two heuristics (CAS_M and CAI_M) not only obtain solutions within 1% from the optimal solutions, but also require no more than 2 seconds of CPU time. In addition, the constructive algorithm CAI forms the initial sequence with a better makespan than CAS. A lower bound, as shown in Theorem 4.3, provides a good insight of the optimal value especially for the cases when the sum of processing times is either much greater than or much less than the sum of loading and unloading times.

An extension of a T-line machining center is studied where the number of CNC machines is increased to two. This problem can be denoted as $F_2 \mid synmv, re-LU \mid C_{\max}$. A dynamic programming algorithm is also developed to obtain the optimal solution for the problem in a small or medium size. Two-phase heuristic algorithms (CAGG_M and CAGI_M) are proposed to obtain a good solution efficiently for a large-size problem.

Similar to the experimental design for the one-machine problem, two constructive algorithms (CAGG and CAGI) are evaluated and the experimental results show that CAGI which employs the insertion scheme performances better than CAGG. The modified neighborhood search applied in the improvement stage also yields significant improvement on the makespan value for the two-machine problem. The makespan value generated by algorithm CAGI_M is on average within 2.3% from the optimal value. The algorithm with the two-phase structure is applicable to a T-line machining center problem with one machine as well as with two machines. In addition, the lower bound derived from Theorem 5.1 is 4.6% from the optimal makespan for the worse case and 6.25% from the solution obtained by algorithm CAGI_M. Therefore, the suggested lower bound is useful when the optimum is not available.

Last but not least, the generalized dynamic programming formulation is derived for the T-line machining scheduling problem with m machines. Moreover, the computational analysis for the generalized dynamic programming algorithm is also conducted and it shows that the complexity of the algorithm is $O(\frac{n!m}{(n-m-2)!}2^{n-m-2})$. For this generalized problem, an algorithm to generate a lower bound is provided, as shown in Theorem 5.2.

6.2 Future Research

The makespan problem of scheduling a two-machine flow shop with one transporter is strongly NP-hard, even when the capacity of the transporter is equal to one and the travel time for the transporter from the first machine to the second machine is equal to the returning time. A heuristic algorithm could be developed to solve the problem with a general setting, such as more than one transporter and a predefined capacity of transporters. In addition, the loading time of a job on the transporter at the first machine and the unloading from the transporter at the second machine can be considered separate from the transportation times. This is because the number of jobs in each batch carried by the transporter may be different. Also, completed jobs can start to be loaded on the transporter when the transporter is waiting for more jobs at the first machine. Regarding the transportation times, non-constant or stochastic transportation times can be a practical extension to the model when the two-machine model is regarded as a two-tier model in a supply chain, for example, a supplier to a manufacturer or a manufacturer to a distributor. In this two-tier model, jobs in the flow shop can be regarded as orders and these orders are delivered by transporters.

In a T-line machining center with two CNC machines, a heuristic algorithm is developed for general configurations of this manufacturing cell. Therefore, to obtain optimality conditions or particular properties for special cases is one of promising research directions. For example, assume the unloading times to be zero, then the problem becomes three-machine flow shop problem with synchronous transfer which has been studied by Soyleu *et al.* (2007). The complexity of this problem can be further investigated. Moreover, a problem with constant loading and unloading times is also similar to the three-machine flow shop problem with synchronous transfer, but the first and last two cycles should be addressed with additional considerations.

Another special case in a two-machine T-line machining center could assume that the processing times of one machine dominate the processing times of the other machine. The assumption means the minimum processing times of one machine is greater than or equal to the maximum processing times of the other machine. Due to the mechanism of synchronous material movement, only the maximum operation time constitutes the time period of each cycle. For example, if the processing times on machine 1 dominate the processing times on machine 2, then the operation on machine 2 can be always neglected except the second last cycle (cycle n+2) because its cycle time (C_{n+2}) is equal to $\max\{p_{[n]2}, u_{[n-1]}\}$. Therefore, this special case is similar to the problem with one CNC machine. Hence, the two-phase heuristic algorithm developed in Chapter 4 can be applied to this special case with minor modifications for cycle *n*+2.

For the T-line machining center scheduling problem, a modified neighborhood search algorithm is applied in the improvement stage. The main advantage of this search algorithm is that it is easy to implement. Despite of its simplicity, the quality of solutions obtained by the proposed algorithm is satisfactory within 6.25% of a lower bound for the two-machine case. In future research, metaheuristic algorithms such as tabu search, simulated annealing and genetic algorithms can be adopted to solve these problems. Metaheuristic algorithms may further improve the quality of solutions but they are more difficult to implement. If the scheduling problem of the T-line machining center becomes more complicated, for example, with more than three machines, the metaheuristics could be more accurate.

In this research, the scheduling problem in a T-line machining center does not consider the interaction between this manufacturing cell and other machines. In addition, all jobs are assumed to be available at time zero. In a practical setting, however, jobs could be processed first by upstream machines before they arrive to the T-line machining center and some jobs may not be available at the beginning of the planning horizon. If a T-line machining center is a bottleneck among these machines, both upstream and downstream machines will adjust their production plans based on the job schedule which can be obtained by the proposed algorithm. On the other hand, if a T-line machining center is not a bottleneck, the practical method to apply the proposed algorithm is to run the algorithm dynamically. For example, the scheduling in a T-line machining center will be triggered periodically and only jobs available at the beginning of each planning period will be sequenced. Other scheduling problems for future research could consider different job arrival times and due dates and different objectives (e.g., minimization of maximum cycle time, total tardiness, etc.).

BIBLIOGRAPHY

Allahverdi, A. and J. Mittenthal (1995). "Scheduling on a Two-Machine Flowshop Subject to Random Breakdowns with a Makespan Objective Function," *European Journal of Operational Research*, **81**, pp.376-387.

Aneja, Y. P. and H. Kamoun (1999). "Scheduling of Parts and Robot Activities in a Two Machine Robotic Cell," *Computers and Operations Research*, **26**, pp.297-312.

Anwar, M. F. and R. Nagi (1998). "Integrated Scheduling of Material Handling and Manufacturing Activitiesfor Just-in-Time Production of Complex Assemblies," *International Journal of Production Research*, **36**(3), pp.653-681.

Armentano, V. A. and D. P. Ronconi (1999). "Tabu Search for Total Tardiness Minimization in Flowshop Scheduling Problems," *Computers & Operations Research*, **26**, pp.219-235.

Bagchi, T. P., J. N. D. Gupta and C. Sriskandarajah (2006). "A Review of Tsp Based Approaches for Flowshop Scheduling," *European Journal of Operational Research*, **169**, pp.816-854.

Baker, K. R. (1995). Elements of Sequencing and Scheduling, Kenneth Baker.

Balasubramanian, J. and I. E. Grossmann (2002). "A Novel Branch and Bound Algorithm for Scheduling Flowshop Plants with Uncertain Processing Times," *Computers and Chemical Engineering*, **26**, pp.41-57.

Ben-Daya, M. and M. Al-Fawzan (1998). "A Tabu Search Approach for the Flow Shop Scheduling Problem," *European Journal of Operational Research*, **109**, pp.88-95.

Chen, B., C. A. Glass, C. N. Potts and V. A. Strusevich (1996). "A New Heuristic for Three-Machine Flow Shop Scheduling," *Operations Research*, **44**(6), pp.891-898.

Conway, R. W., W. L. Maxwell and L. W. Miller (1967). Theory of Scheduling. MA, Addison-Wesley: Reading.

Dai, J. G. and G. Weiss (2002). "A Fluid Heuristic for Minimizing Makespan in Job Shops," *Operations Research*, **50**(4), pp.692-707.

Dawande, M., H. N. Geismar, S. P. Sethi and C. Sriskandarajah (2005). "Sequencing and Scheduling in Robotic Cells: Recent Developments," *Journal of Scheduling*, **8**, pp.387-426.

Dell'amico, M. (1996). "Shop Problems with Two Machines and Time Lags," *Operation Research*, **44**(5), pp.777-787.

Dorigo, M. and L. M. Gambardella (1997). "Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, **1**, pp.53-66.

Dreyfus, S. E. and A. M. Law (1997). The Art and Theory of Dynamic Programming. New York, Academic Press.

Framinan, J. M., J. N. D. Gupta and R. Leisten (2004). "A Review and Classification of Heuristics for Permutation Flow-Shop Scheduling with Makespan Objective," *Journal of the Operational Research Society*, **55**, pp.1243-1255.

Garey, M. R. and D. S. Johnson (1979). Computer and Intractability - a Guild to the Theroy of Np-Completeness. New York W. H. Freeman and Company.

Garey, M. R., D. S. Johnson and R. Sethi (1976). "The Complexity of Flow-Shop and Job-Shop Scheduling," *Math. Operation Research*, **1**, pp.117–129.

Gilmore, P. C. and R. E. Gomory (1964). "Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem," *Operation Research*, **12**(5), pp.655-679.

Gonçalves, J. F., J. J. Mendes and M. G. C. Resende (2005). "A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem," *European Journal of Operational Research*, **167**, pp.77-95.

Gourgand, M., N. Grangeon and S. Norre (2003). "A Contribution to the Stochastic Flow Shop Scheduling Problem," *European Journal of Operational Research*, **151**(415-433).

Grabowskia, J. and M. Wodecki (2004). "A Very Fast Tabu Search Algorithm for the Permutation Flow Shop Problem with Makespan Criterion," *Computers & Operations Research*, **31**, pp.1891-1909.

Graham, R. L., E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan (1979). "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Annals of Discrete Mathematics*, **5**, pp.287-326.

Graves, S. C. (1981). "A Review of Production Scheduling," *Operation Research*, **29**(4), pp.646-675.

Gupta, J. N. D. and E. F. Stafford (2006). "Flowshop Scheduling Research after Five Decades," *European Journal of Operational Research*, **169**, pp.699-711.

Hajek, B. (1988). "Cooling Schedules for Optimal Annealing," *Mathematics of Operations Research*, **13**, pp.311-329.

Hall, N. G., H. Kamoun and C. Sriskandarajah (1997). "Scheduling in Robotic Cells: Classification, Two and Three Machine Cells," *Operations Research*, **45**(3), pp.421-439.

Hall, N. G. and C. Sriskandarajah (1996). "A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process," *Operations Research*, **44**(3), pp.510-525.

Han, M.-H. and L. F. Mcginnis (1989). "Control of Material Handling Transporter in Automated Manufacturing," *IIE Transactions*, **2**, pp.184-190.

Hejazi, S. R. and S. Saghafian (2005). "Flowshop-Scheduling Problems with Makespan Criterion: A Review," *Journal of Production Research*, **43**(14), pp.2895-2929.

Hurink, J. and S. Knust (2001). "Makespan Minimization for Flow-Shop Problems with Transportation Times and a Single Robot," *Discrete Applied Mathematics*, pp.199-216.

Hurink, J. and S. Knust (2002). "A Tabu Search Algorithm for Scheduling a Single Robot in a Job-Shop Environment," *Discrete Applied Mathematics*, **119**, pp.181-203.

Jain, A. S. and S. Meeran (1999). "Deterministic Job-Shop Scheduling: Past, Present and Future," *European Journal of Operational Research*, **113**, pp.390-434.

Johnson, S. M. (1954). "Optimal Two- and Three-Stage Production Schedules with Setup Times Included," *Naval Research Logistics Quarterly*, **1**, pp.61-68.

Kalczynski, P. J. and J. Kamburowski (2006). "A Heuristic for Minimizing the Expected Makespan in Two-Machine Flow Shops with Consistent Coefficients of Variation," *European Journal of Operational Research*, **169**, pp.742-750.

Kamburowski, J. (1999). "Stochastically Minimizing the Makespan in Two-Machine Flow Shops without Blocking," *European Journal of Operational Research*, **112**, pp.304-309.

Karuno, Y. and H. Nagamochi (2003). A Better Approximation for the Two-Machine Flowshop Scheduling Problem with Time Lags. The 14th Annual International Symposium on Algorithms and Computation. Kyoto**:** 309-318.

Kijima, M., N. Makimoto and H. Shirakawa (1990). "Stochastic Minimization of the Makespan in Flow Shops with Identical Machines and Buffers of Arbitrary Size," *Operations Research*, **38**(5), pp.924-928.

Kis, T. (2003). "Jop-Shop Scheduling with Processing Alternatives," *European Journal of Operational Research*, **151**, pp.307-332.

Lai, T.-C. (1996). "A Note on Heuristics of Flow-Shop Scheduling," *Operations Research*, **44**(4), pp.648-652.

Lee, C.-Y. and Z.-L. Chen (2001). "Machine Scheduling with Transportation Considerations," *Journal of Scheduling*, pp.3-24.

Lee, C.-Y., L. Lei and M. Pinedo (1997). "Current Trends in Deterministic Scheduling," *Annals of Operation Research*, **70**, pp.1-41.

Lee, C.-Y. and V. A. Strusevich (2005). "Two-Machine Shop Scheduling with an Uncapacitated Interstage Transporter," *IIE Transactions*, **37**, pp.725-736.

Lenstra, J. K. and K. Rinnooy (1979). "Computational Complexity of Discrete Optimization Problems," *Annals of Discrete Mathematics*, **4**, pp.121-140.

Leung, J. Y.-T. (2004). Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press.

Levner, E., K. Kogan and I. Levin (1995). "Scheduling a Two-Machine Robotic Cell: A Solvable Case," *Annals of Operations Research*, **5**, pp.217-232.

Levner, E., K. Kogan and O. Maimon (1995). "Flowshop Scheduling of Robotic Cells with Job-Dependent Transportation and Set-up Effects," *Journal of Operational Research Society,*, **46**(12), pp.1447-1455.

Logendran, R. and C. Srisjandarajah (1996). "Sequencing of Robot Activities and Parts in Two-Machine Robotic Cells," *International Journal of Production Research*, **34**(12), pp.3447-3463.

Logendran, R. and C. Sriskandarajah (1993). "Two-Machine Group Scheduling Problem with Blocking and Anticipatory Setups," *European Journal of Operational Research*, **69**(3), pp.467-481.

Low, C. (2005). "Simulated Annealing Heuristic for Flow Shop Scheduling Problems with Unrelated Parallel Machines," *Computers & Operations Research*, **32**, pp.2013-2025.

Mattfeld, D. C. and C. Bierwirth (2004). "An Efficient Genetic Algorithm for Job Shop Scheduling with Tardiness Objectives," *European Journal of Operational Research*, **155**, pp.616-630.

Meyers, F. E. and M. P. Stephens (2005). Manufactuing Facility Design and Material Handling. New Jersey, Pearson Prentice Hall.

Milacron, C. (1989). "T-Line Machining Center Alternatives," *Manufacturing Engineering*, **103**(4), pp.14-15.

Nowicki, E. and C. Smutnicki (2005). "An Advanced Tabu Search Algorithm for the Job Shop Problem," *Journal of Scheduling*, **8**, pp.145-159.

Oulamara, A. and A. Soukhal (2001). Flow Shop Scheduling Problems with Transportation and Capacities Constraints. IEEE International Conference on Systems, Man, and Cybernetics. **4:** 2540-2545.

Panwalkar, S. S. and W. Iskander (1977). "A Survery of Scheduling Rules," *Operation Research*, **25**(1), pp.45-61.

Pezzella, F. and E. Merelli (2000). "A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem," *European Journal of Operational Research*, **120**, pp.297-310.

Pinedo, M. (1983). "Stochastic Scheduling with Release Dates and Due Dates," *Operations Research*, **31**(3), pp.559-572.

Pinedo, M. (1985). "A Note on Stochastic Shop Models in Which Jobs Have the Same Processing Requirements on Each Machine," *Management Science*, **31**(7), pp.840-846.

Pinedo, M. (1995). Scheduling : Theory, Algorithms, and Systems Prentice Hall.

Ponnambalam, S. G., P. Aravindan and S. Chandrasekaran (2001). "Constructive and Improvement Flow Shop Scheduling Heuristics: An Extensive Evaluation," *Production Planning and Control*, **12**(4), pp.335-344.

Ponnambalam, S. G., P. Aravindan and S. V. Rajesh (2000). "A Tabu Search Algorithm for Job Shop Scheduling," *International Journal of Advanced Manufacturing Technology*, **16**, pp.765-771.

Rajendran, C. and H. Ziegler (2004). "Ant-Colony Algorithms for Permutation Flowshop Scheduling to Minimize Makespan/Total Flowtime of Jobs," *European Journal of Operational Research*, **155**, pp.426-438.

Rebaine, D. and V. A. Strusevich (1999). "Two-Machine Open Shop Scheduling with Special Transportation Times," *Journal of the Operational Research Society*, **50**, pp.756-764.

Reeves, C. R. (1995). "A Genetic Algorithm for Flowshop Sequencing," *Comupters and Operations Research*, **22**(1), pp.5-13.

Ruiz, R. and C. Maroto (2005). "A Comprehensive Review and Evaluation of Permutation Flowshop Heuristics," *European Journal of Operational Research*, **165**, pp.479-494.

Schutten, J. M. J. (1998). "Practical Job Shop Scheduling," *Annals of Operation Research*, **83**, pp.161-177.

Sethi, S. P., C. Srisjandarajah, G. Sorger, J. Blazewicz and W. Kubiak (1992). "Sequencing of Parts and Robot Moves in a Robotic Cell," *International Journal of Flexible Manufacturing Systems*, **4**, pp.331-358.

Soukhal, A., A. Oulamara and P. Martineau (2005). "Complexity of Flow Shop Scheduling Problems with Transportation Constraints," *European Journal of Operational Research*, **161**, pp.32-41.

Soyleu, B., . Kirca and M. Azizoğlu (2007). "Flow Shop-Sequencing Problem with Synchronous Transfers and Makespan Minimization," *Internationals Journal of Production Research*, **45**(15), pp.3311-3331.

Strusevich, V. A. (1999). "A Heuristic for the Two-Machine Open-Shop Scheduling Problem with Transportation Times," *Discrete Applied Mathematics*, **93**, pp.287-304.

Szwarc, W. (1983). "Flow Shop Problems with Time Lags," *Management Science*, **29**(4), pp.477-481.

Taillard, E. (1993). "Benchmarks for Basic Scheduling Problems," *European Journal of Operational Research*, **64**, pp.278–285.

Talwar, P. P. (1967). "A Note on Sequencing Problems with Uncertain Job Times," *Journal of Operations Research Society Japan* **9**, pp.93-97.

Tian, P., J. Ma and D.-M. Zhang (1999). "Application of the Simulated Annealing Algorithm to the Combinatorial Optimisation Problem with Permutation Property: An Investigation of Generation Mechanism," *European Journal of Operational Research*, **118**, pp.81-94.

Tompkin, J. A. and J. A. White (1984). Facilities Planning. New York, Wiley.

Wang, L. and L. Zhang (2006). "Stochastic Optimization Using Simulated Annealing with Hypothesis Test," *Applied Mathematics and Computation*, **174**(2), pp.1329-1342.

Wang, L., L. Zhang and D.-Z. Zheng (2003). "A Class of Order-Based Genetic Algorithm for Flow Shop Scheduling," *International Journal of Advanced Manufacturing Technology*, **22**, pp.828-835.

Wang, L. and D.-Z. Zheng (2001). "An Effective Hybrid Optimization Strategy for Job-Shop Scheduling Problems," *Computers and Operations Research*, **28**, pp.585-596.

Wang, L. and D.-Z. Zheng (2003). "An Effective Hybrid Heuristic for Flow Shop Scheduling," *International Journal of Advanced Manufacturing Technology*, **21**, pp.38-44.

Ying, K.-C. and C.-J. Liao (2004). "An Ant Colony System for Permutation &Ow-Shop Sequencing," *Computers & Operations Research*, **31**, pp.791-801.

Yu, W. (1996). "The Two-Machine Flow Shop Problem with Delays and the One-Machine Total Tardiness Problem", Thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven

Yu, W., H. Hoogeveen and J. K. Lenstra (2004). "Minimizing Makespan in a Two-Machine Flow Shop with Delays and Uni-Time Operations Is Np-Hard," *Journal of Scheduling*, **7**, pp.333-348.

Zegordi, S. H., K. Itoh and T. Enkawa (1995). "Minimizing Makespan for Flow Shop Scheduling by Combining Simulated Annealing with Sequencing Knowledge," *European Journal of Operational Research*, **85**, pp.515-531.

APPENDIX

Gilmore-Gomory Algorithm

The Gilmore and Gomory algorithm is widely implemented in literature and its procedure can be found in these papers such as Gilmore and Gomory (1964), Hall and Sriskandarajah (1996) and Bagchi *et al.*(2006).

Step 1. Sort $\{p_{j2}\}$ in the non-decreasing order and renumber the jobs in the order such that $p_{j2} \leq p_{j+1,2}, j = 1,\ldots,n-1$. Initialize $G_1 = G_2 = \varnothing$.

Step 2. Sort $\{p_{j1}\}$ in the non-decreasing order and define a variable $\phi(j)$ for every job such that $p_{\phi(j)1} \leq p_{\phi(j+1)2}, j = 1,\ldots,n-1$.

Step 3. Calculate the cost of an edge as

$C_{j,j+1} = \max\{0,\ (\min\{p_{j+1,2},\ p_{\phi(j+1)1}\} - \max\{p_{j2},\ p_{\phi(j)1}\})\}$ for $j = 1,\ldots,n-1$

Step 4. Construct the graph with undirected edges $(j,\ \phi(j))$ from $j = 1$ to n.

Step 4.1. Add the undirected edge $(j,\ \phi(j))$ which is not in the graph and $j \neq \phi(j)$. Set $j = j+1$. Repeat the step until $j > n$.

Step 5. If the current graph has only one connected component, go to Step 7. Otherwise, connect the graph to be one component as the following procedure:

Step 5.1. Sort $C_{j,j+1},\ j = 1,\ldots,n-1$ in non-decreasing order.

Step 5.2. For $j = 1$ to $n-1$. If the edge $(j,\ j+1)$ with j and $j+1$ in different components, add the edge to the graph. Otherwise go to Step 5.4.

Step 5.3. If $p_{1,\phi(j)} \geq p_{2,j}$, set $G_1 = G_1 \cup \{(j,\ j+1)\}$. Otherwise, set $G_2 = G_2 \cup \{(j,\ j+1)\}$.

Step 5.4. If all nodes are connected, go to Step 6. Otherwise, set $j = j+1$ and go to Step5.2.1.

Step 6. If $G_1 \neq \varnothing$, order the edges in G_1 as $\{(r_1, r_1+1), \ldots, (r_g, r_g+1)\}$, where $r_1 \geq \ldots \geq r_g$ and $g = |G_1|$.

Step 7. If $G_2 = \varnothing$, order the edges in G_2 as $\{(s_1, s_1+1), \ldots, (s_h, s_h+1)\}$, where $s_1 \leq \ldots \leq s_h$ and $h = |G_2|$.

Step 8. Form the tour with the minimum cost by finding the job follows job *j*. Define a function $\alpha_{p,q}(j)$ as follows: $\alpha_{p,q}(j) = q$ if $j = p$, $\alpha_{p,q}(j) = p$ if $j = q$, and $\alpha_{p,q}(j) = j$ if $j \neq p, q$ where $j, p, q = 1, \ldots, n$.

Step 8.1. Define $\psi(j) = \phi(\alpha_{r_1, r_1+1}\alpha_{r_2, r_2+1} \cdots \alpha_{r_{|G_1|}, r_{|G_1|}+1}\alpha_{s_1, s_1+1}\alpha_{s_2, s_2+1} \cdots \alpha_{s_{|G_2|}, s_{|G_2|}+1}(j))$. For $j = 1, \ldots, n$, to obtain job $\psi(j)$ which follows job *j*.

A Numerical Example

A numerical example is presented to illustrate the procedure of the algorithm. In this example, there are 8 jobs and its processing times are summarized in Table A.1: The objective is to find a tour with the minimum cost.

Table A.1: Job data for the Gilmore-Gomory Algorithm

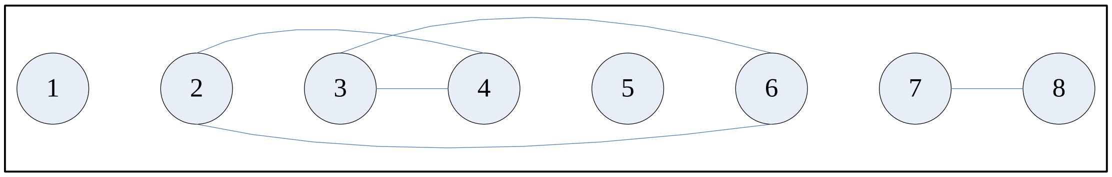
Job	1	2	3	4	5	6	7	8
p_{j1}	7	3	15	8	10	12	13	17
p_{j2}	14	1	19	9	5	13	8	16

Step 1, Step 2 and Step 3 give Table A.2:

Table A.2: Sorted processing times and the cost for each edge

j	p_{j2}	$p_{\phi(j)1}$	$\phi(j)$	$\max\{p_{j2}, p_{\phi(j)1}\}$	$\min\{p_{j2}, p_{\phi(j)1}\}$	$C_{j,j+1}$
1	1	3	1	3	1	2
2	5	7	6	7	5	1
3	8	8	4	8	8	1
4	9	10	2	10	9	2
5	13	12	5	13	12	0
6	14	13	3	14	13	1
7	16	15	8	16	15	1
8	19	17	7	19	17	–

Step 4. The graph with undirected edges $(j,\ \phi(j))$.



Step 5.

$$G_1 = \{(1,2)\}$$

$$G_2 = \{(5,6),(6,7)\}$$

Step 6.

$$G_1 = \{(1,\ 2)\},\ r_1 = 1$$

Step 7.

$$G_2 = \{(5,6),(6,7)\},\ s_1 \le s_2 \Rightarrow\ s_1 = 5, s_2 = 6$$

Step 8.

$$\psi(j) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(j))$$

$$j = 1:\ \ \psi(1) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(1)) = \phi(\alpha_{1,2}\alpha_{5,6}(1)) = \phi(\alpha_{1,2}(1)) = \phi(2) = 6$$

$$j = 2: \ \psi(2) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(2)) = \phi(\alpha_{1,2}\alpha_{5,6}(2)) = \phi(\alpha_{1,2}(2)) = \phi(1) = 1$$

$$j = 3: \ \psi(3) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(3)) = \phi(\alpha_{1,2}\alpha_{5,6}(3)) = \phi(\alpha_{1,2}(3)) = \phi(3) = 4$$

$$j = 4: \ \psi(4) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(4)) = \phi(\alpha_{1,2}\alpha_{5,6}(4)) = \phi(\alpha_{1,2}(4)) = \phi(4) = 2$$

$$j = 5: \ \psi(5) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(5)) = \phi(\alpha_{1,2}\alpha_{5,6}(5)) = \phi(\alpha_{1,2}(6)) = \phi(6) = 3$$

$$j = 6: \ \psi(6) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(6)) = \phi(\alpha_{1,2}\alpha_{5,6}(7)) = \phi(\alpha_{1,2}(7)) = \phi(7) = 8$$

$$j = 7: \ \psi(7) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(7)) = \phi(\alpha_{1,2}\alpha_{5,6}(6)) = \phi(\alpha_{1,2}(5)) = \phi(5) = 5$$

$$j = 8: \ \psi(8) = \phi(\alpha_{1,2}\alpha_{5,6}\alpha_{6,7}(8)) = \phi(\alpha_{1,2}\alpha_{5,6}(8)) = \phi(\alpha_{1,2}(8)) = \phi(8) = 7$$

Thus, the optimal tour is $1-6-8-7-5-3-4-2$ and the minimum cost is 94. Table A.3 is the summary of each job is followed by job $\psi(j)$.

Table A.3: The optimal tour

Job	1	2	3	4	5	6	7	8
$\psi(j)$	6	1	4	2	3	8	5	7

VITA

Kwei-Long Huang

Kwei-Long Huang was born in Nantou, Taiwan, on May 20, 1975. He received Bachelor and Master of Business Administration (BBA and MBA) degrees in Information Management from National Taiwan University in 1997 and 1999, respectively. After completing the military service, he worked for AsiaTEK as a systems analyst to develop an Advanced Planning and Scheduling (APS) system. In 2002, he transferred to Compal Electronics as a SAP engineer in the Information Technology department. He joined the Ph.D. program in Industrial and Manufacturing Engineering at the Pennsylvania State University in 2004. His research interests are in the areas of production planning and scheduling, supply chain management, and mathematical programming.