國立臺灣大學管理學院資訊管理學研究所

碩士論文

Department of Information Management
College of Management
National Taiwan University

Master's Thesis

考慮等候時間上限與限期完成之保養的彈性零工式排程問題

A Flexible Job Shop Scheduling Problem with Required Maintenances Considering Queue Time Limits

龔雪燕

Hsueh-Yen Kung

指導教授：孔令傑 博士

Advisor: Ling-Chieh Kung, Ph.D.

中華民國 113 年 1 月

January 2024

doi:10.6342/NTU202400496

國立臺灣大學碩士學位論文
口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

考慮等候時間上限與限期完成之保養的

彈性零工式排程問題

A Flexible Job Shop Scheduling Problem with Required Maintenances Considering Queue Time Limits

本論文係龔雪燕（R10725043）在國立臺灣大學資訊管理學研究所論文，於民國 112 年 07 月 10 日承下列考試委員審查通過及口試及格，特此證明。

口試委員 Oral examination committee:

邱舍儒		
（指導教授 Advisor）		
黃奇隆	藍俊宏	吳玠潔

系主任/所長 Director: 陳彥賓

謝辭

研究所對我來說並不是理所當然的計畫，過程中受到了很多支持與指引，才能夠有今日的收穫，感謝一路上所遇到的人事物，成就了現在的自己。兩年的時間很短，每分每秒都顯得彌足珍貴，感謝指導老師小傑總會在我們感到不知所措與迷茫時給予建議，面對研究這個龐大的課題，因為有小傑老師的耐心指導和關懷，才能讓我順利完成任務，另外也在老師的帶領下參與了產學專案、研討會和課程助教，除了專業技能以外，也學習到處事的態度和方法，非常感謝小傑老師的指導。除此之外，也非常感謝黃奎隆老師、藍俊宏老師和吳政鴻老師的指導，讓我的論文更加完整。這兩年也非常幸運能認識 IEDO Lab 的大家。感謝同屆的夥伴一起闖蕩研究所的各種難關。謝謝若瑜陪伴了我研究所各種時刻，許多的難關都因為有你而變得渺小，也謝謝仲嘉、恩淇和楚軒，常常分享許多有趣的事和觀點，每次都能給我不同的感觸。謝謝可愛的學姊們讓剛踏入研究所的我獲得滿滿的溫暖，許多窩心時刻還有和 PO Lab 一起的活動，都為我的研究所生活增添了許多色彩。也謝謝貼心的學弟妹，陪伴著我們的研究時光，和我們一起討論論文還有各式各樣的課程，謝謝琳瑄和元婷總是會記得許多大大小小的日子，在顛頗的生活裡變出驚喜，給我們滿滿的鼓勵。在此，也想感謝我的大學專題老師康藝晃，鼓勵我繼續就讀研究所，讓我能有這段寶貴的經歷與更寬廣的眼界。也謝謝卓雍然老師的耐心指導，給我許多表現和學習的機會，讓我可以在跨領域的過程中有更多的成長。最後，我想感謝一直在背後默默支持我的家人，謝謝你們一直以來的陪伴，有你們的支持和鼓勵，我才能是今天的我。感謝一直在我身邊的所有人，也期許自己未來能成為一個處事從容、待人溫暖且獨特的人。

龔雪燕

于臺灣大學資訊管理學研究所

民國一百一十二年七月

doi:10.6342/NTU202400496

摘要

工作生產排程和機台保養之間的權衡取捨對於解決排程問題至關重要。當暫停生產工作進行機台維護時，使得生產的排程延後，也可能使得預定好的訂單無法如期交貨。從另一方面來看，如果持續進行生產而輕忽機台的保養維護，機台可能會嚴重耗損，進而影響產品品質，也可能有無法預期的災難性損失出現，其中也包含了嚴重的交期延誤。因此，本研究針對生產排程與保養排程的聯合調度問題進行探討，聚焦在彈性零工式的生產排程並考慮一個工作可能會通過一個工作站數次的情境。在我們的問題設計上，需要被維修的機台都是已知的，在指定期限前必須完成指定時間長度的保養，但保養的開始時間點則由決策者決定。我們的問題會同時考慮每個製程步驟之間的等候時間上限。在上述情境設定下，本研究目標為最小化加權後的總延遲時間。

由於混合整數規劃模型無法在合理的時間內找到最佳解，我們提出了一種基於貪婪演算法延伸的啟發式演算法來解決我們的問題，同時我們也使用了基因演算法來提升演算法的表現。為了證明此啟發式演算法的有效性，我們在四種環境設定七種情境的實驗來檢驗演算法的表現。結果顯示，我們的演算法在工作不會經過重複製程步驟的情況下表現較佳，並且可以有效地減少計算時間。最後，我們也使用台灣面板公司提供的資料透過我們提出的演算法進行排程，結果也顯示了我們的驗算法在實務上的可用性。

關鍵字： *零工式排程、預防性保養排程、混合整數規劃、等候時間限制、啟發性演算法*

doi:10.6342/NTU202400496

Abstract

A balance between production scheduling and machine maintenance is crucial. Halting production temporarily for maintenance can cause schedule delays. However, neglecting machine maintenance can result in significant wear and tear, affecting product quality and potentially leading to unforeseen catastrophic losses. Therefore, this study aims to optimize production and maintenance scheduling in a job shop and consider the scenario where a job may pass through a specific station multiple times. We assume that the due times for completing fixed-length maintenance on some machines are given, and the planner may determine when to start each maintenance. The objective is to minimize total weighted tardiness.

We propose a heuristic algorithm based on the greedy algorithm because the mixed integer programming model may fail to find the optimal solution within a reasonable time. Additionally, we use the Genetic Algorithm to enhance the performance of the algorithm. To demonstrate the effectiveness, we conduct experiments in four different environmental situations with seven scenarios to evaluate its performance. The results show that our algorithm performs better with no recirculation situation and significantly reduces computation time. Furthermore, we apply it to a real-world case of a manufacturer in Taiwan, and the experimental results confirm the applicability of the algorithm in the practical scenarios.

Keywords: *job shop scheduling*, *preventive maintenance*, *mixed integer programming*, *queue time limit*, *heuristic algorithm*

doi:10.6342/NTU202400496

Contents

doi:10.6342/NTU202400496

doi:10.6342/NTU202400496

doi:10.6342/NTU202400496

List of Figures

doi:10.6342/NTU202400496

doi:10.6342/NTU202400496

List of Tables

doi:10.6342/NTU202400496

doi:10.6342/NTU202400496

Chapter 1

Introduction

1.1 Background and motivation

Nowadays, in the real world, many factories are capable of monitoring machine conditions and obtaining real-time data from modern sensors to promptly identify machines that require maintenance in the near future. However, due to the interconnectedness of maintenance and production plans, the production environment becomes highly complicated. Even if we identify the machines in need of maintenance, determining the optimal timing for maintenance remains a challenge.

The scheduling problem holds significant importance in various industries as it enables companies and factories to allocate resources and devise production plans. Within the production industry, optimizing the production process is a critical concern. By implementing an optimal schedule, companies can reduce costs associated with human resources, operations, and production, all while maintaining the existing production plan.

doi:10.6342/NTU202400496

Furthermore, there are often various additional tasks that need to be performed on the production floor, such as machine maintenance. Based on the machine's operating condition, such as elapsed time, the factory designs a maintenance plan. This type of maintenance is referred to as *preventive maintenance*, which falls under one of the two main maintenance categories, the other being *corrective maintenance* (Wang, 2002). Maintenance activities are essential for machine operations; neglecting them may lead to unforeseen damage or downtime. Therefore, in this study, we consider the insertion task as a required machine maintenance activity.

At the production site, insertion maintenance may be scheduled based on on-site conditions and staff experience. However, due to the complicatedity of the production process, it is challenging to evaluate it accurately. If the on-site personnel rely solely on their experience to schedule maintenance work, it could potentially disrupt the original production plan, leading to negative consequences for the company. These consequences may include a loss of customer trust, increased production and labor costs, and delayed order fulfillment.

Moreover, when maintenance needs to be inserted into the predetermined production schedule, it inevitably interrupts the ongoing production. To achieve optimal productivity and minimize the impact of insertion maintenance on the production schedule, it is crucial to plan both production and maintenance schedules simultaneously.

In our study, we are considering a flexible job shop scheduling problem that involves required maintenance on selected machines. One prominent example of such a production scheduling environment is the production of TFT-LCD (Thin Film Transistor - Liquid Crystal Display) panels.

doi:10.6342/NTU202400496

Furthermore, another crucial aspect of the TFT-LCD production environment is the limitation on queue time between job steps. If the queue time exceeds a certain limit, it can lead to job rework, resulting in increased time and production costs. In our study, we need to consider the maintenance plan simultaneously, but inserting maintenance into the production schedule may result in excessively long queue time between the steps, leading to the over-queue-time issue. Therefore, one of the key focuses of this study is to explore how to prevent the occurrence of over-queue-time issue.

While previous research has delved into the flexible job shop scheduling problem, there is limited literature specifically addressing the constraints of queue time, the re-entrant environment, and the required maintenance plans. Nevertheless, these factors have a significant impact on machine availability and can potentially affect the timeliness of production orders. Therefore, we have constructed a model to address this issue in our problem: the flexible job shop scheduling problem in the re-entrant environment with queue time limits and required machine maintenance plans. The objective of our model is to minimize the total weighted tardiness.

1.2 Research objectives

For our study, we are considering the flexible job shop scheduling problem in the re-entrant environment with a queue time constraint and the requirement for machine maintenance activities. Before the scheduling process begins, a decision maker is responsible for generating an executable schedule of jobs and maintenance tasks. Another planner will monitor the machine conditions and determine which machines need to be

doi:10.6342/NTU202400496

maintained at least once in the upcoming days.

Based on the available information, our decision maker will schedule both jobs and maintenance activities simultaneously, aiming to minimize the total weighted tardiness of the production. Within the planning horizon, the jobs in the production plan must be completed, and the machines that require maintenance must be serviced exactly once before a given due time.

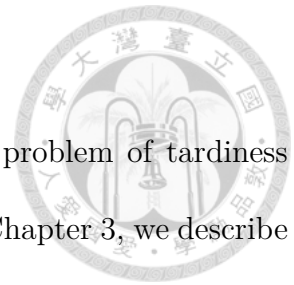
In our flexible job shop scheduling problem under the re-entrant environment, each job from the production plan needs to go through multiple steps, which are performed by various processes. Specifically, a process consists of multiple steps that must be processed on the same group of machines. These groups of machines are referred to as stations, and re-entrant occurs in the production environment. Furthermore, each job has different release times, due times, and weights of tardiness. Additionally, each job at different steps has distinct queue time limits and processing times.

Each machine can only handle one job or one maintenance task at a time. When a machine requires maintenance, some jobs may be postponed, resulting in tardiness. Our model aims to determine both the production and maintenance schedules while considering the queue time limit in order to minimize the total weighted tardiness of orders.

Since the one-machine scheduling problem, which involves minimizing total tardiness, is considered an NP-hard problem (Du and Leung, 1990), it is believed to be unsolvable in polynomial time. Moreover, our problem is even more complicated than the one-machine scheduling problem. To enhance the model's usability in a production setting, we propose a heuristic algorithm to find an approximate best solution within a reasonable time frame.

doi:10.6342/NTU202400496

1.3 Research plan



Next chapter, we will review the literature about the scheduling problem of tardiness minimization, and also compare the differences from our study. In Chapter 3, we describe the model of our problem. This model is formulated by MIP and considers the flexible job shop scheduling problem in the re-entrant environment with the queue time constraint and required maintenance.

Based on the problem formulation, our heuristic algorithm is proposed in Chapter 4. The numerical experiments' performance and time efficiency of the MIP model and heuristic algorithm are displayed in Chapter 5. In Chapter 6, a case study with the panel company is described. Finally, our conclusions of this study are described in Chapter 7.

doi:10.6342/NTU202400496

Chapter 2

Literature Review

2.1 Tardiness minimization for a single stage scheduling problem

Recently, the criteria for order tardiness have become more important than makespan in industrial plants (Vallada et al., 2008). There is an increasing number of studies focusing on minimizing tardiness in different scheduling problems. This objective has gained significant attention in real-life scenarios.

Abdul-Razaq et al. (1990) survey algorithms for the single machine scheduling problem to minimize the total weighted tardiness. They consider various algorithms, including dynamic programming and branch-and-bound algorithms.

Furthermore, there is literature that investigates tardiness minimization in parallel machine environments. Yalaoui and Chu (2002) employ the branch-and-bound algorithm to solve the problem, and Biskup et al. (2008) develop efficient heuristic algorithms to find

doi:10.6342/NTU202400496

optimal solutions for small cases or near-optimal schedules for large cases. In the case of unrelated parallel problems, Liaw et al. (2003) not only utilize the branch-and-bound algorithm but also incorporated various dominance rules into the algorithm. Additionally, Chen and Wu (2006) develop an effective heuristic based on threshold-accepting methods, improvement procedures, and tabu lists, considering setup costs if the production process incurred die exchange. Moreover, de Alba et al. (2022) propose a new MIP model using linear positional variables instead of the Big-M technique and presented an efficient iterated local search algorithm to solve the problem within reasonable timeframes.

The settings of the machines in the above researches are either single machines or single stages with parallel machines. However, in our problem setting, the orders consist of multiple steps and require parallel machines. In the next section, we will review the flow shop scheduling problem, which is a scheduling problem with multiple steps.

2.2 Tardiness minimization in the flow shop scheduling problem

Some research studies the flow shop scheduling problem with the two-stage situation. Choi and Lee (2009) consider a static scenario where they assumed that the release time of all jobs is at time zero, the due dates are deterministic and given in advance, and the job processing times are the same for the parallel machines at each stage. They employ the branch-and-bound algorithm and two-phase heuristic algorithms to solve the two-stage hybrid flow shop scheduling problem, which involves multiple identical parallel machines in each production stage. The branch-and-bound algorithm provides optimal solutions

doi:10.6342/NTU202400496

for small problems, while the two-phase heuristic algorithms are used for larger problems. Allahverdi and Aydilek (2015) study the two-stage flow shop scheduling problem using multiple algorithms, including an insertion algorithm, a genetic algorithm, two versions of simulated annealing algorithm, and two versions of cloud theory-based simulated annealing algorithm. However, the second stage setting only involves one machine and only handles the final operation of jobs. Yu et al. (2017) study the batching with two-stage hybrid flow shops problem, in which each job has a distinct due date and two operations. They assume identical parallel machines in each production stage. Each batch contains a group of jobs, but these jobs are processed individually. Once the batches have been processed on one of the parallel machines in the initial stage, they are moved to the second stage without being split and then processed on one of the parallel machines. This study aims to minimize the total job tardiness and investigates various factors, including the number of batches, the composition of each batch, the assignment of batches to parallel machines at each stage, and the sequence of batches allocated to each machine. Furthermore, under the static and deterministic setting, the study develops a MIP model with a predetermined number of batches and proposes three iterative algorithms to solve the problem.

For the multiple stage problem, Liao and Huang (2010) study the flow shop scheduling problem with non-permutation, allowing different job sequences on different machines. This study proposed three MILP models and two Tabu search-based algorithms to solve the problem.

There are some differences between the aforementioned flow-shop literature and our work. First, our study considers the re-entrant environment, some steps of the job are

doi:10.6342/NTU202400496

processed by the same station. Second, to align with the production procedure, we considered a queue time constraint between job steps. In the TFT-LCD industry, the queue time between steps must not exceed the predefined limit; otherwise, the job needs to be reworked, resulting in increased redundant production costs. Third, our study incorporates a special type of job known as maintenance tasks. These tasks must be scheduled within the planning horizon to avoid unexpected risks. Additionally, due to limited maintenance resources, the execution times of these tasks should not overlap. These settings not only make the problem more realistic but also increase its complexity and difficulty.

2.3 Scheduling problem with queue time constraints

Queue time control is a critical issue in the TFT-LCD industry. Queue time constraints impose upper bounds on the processing time of each job at different steps, ensuring that the time elapsed between a step and its previous step does not exceed its corresponding limit. These upper bounds are referred to as queue time limits. If the elapsed time exceeds the queue time limit, it can lead to decreased production quality and loss of production capacity.

In their work, Yang and Chern (1995) address a two-machine flow shop sequencing problem with limited waiting time constraints, aiming to minimize the makespan. They propose a branch-and-bound algorithm to solve the problem effectively. On the other hand, Su (2003) tackle a hybrid two-stage flow shop with a batch processor in the first stage and a single processor in the second stage, also aiming to minimize the makespan.

doi:10.6342/NTU202400496

They develop a mixed integer programming model as a benchmark and proposed a heuristic algorithm for solving the problem efficiently.

Akkerman et al. (2007) investigate capacity-constrained and time-constrained intermediate storage in two-stage food production systems. The time-constrained aspect aims to prevent spoilage of unpackaged products. To ensure timely delivery, products must be packaged and transported within a specific time frame; otherwise, they must be discarded as waste or low-quality by-products. This study focuses on various performance measures, such as flow time, makespan, and waste, and employs simulation techniques to examine system behavior under different capacities and time constraints on intermediate storage. The numerical experiments reveal that the time constraint on storage significantly impacts the performance of the production system.

Chen and Tang (2012) address flow shop scheduling problems with re-entrant flows and queue time constraints. In re-entrant flow environments, a job may need to revisit a particular station multiple times for additional operations. The objective function aims to minimize the number of tardy jobs, and a rule-based heuristic algorithm is proposed to achieve this.

To the best of our knowledge, the scheduling problem study considers the queue time limitation in the re-entrant environment, but it has not addressed the required machine maintenance so far. Therefore, we propose the MIP model and the heuristic algorithm to address this issue.

Based on the aforementioned settings, our model is better equipped to handle complex situations in the production industry, making it more realistic.

doi:10.6342/NTU202400496

Chapter 3

Problem Description and Formulation

3.1 Problem description

In this section, we describe the setting of the joint preventive maintenance and job scheduling problem in the flexible job shop environment and present the problem formulation.

At the beginning of the planning horizon, the decision maker is given a required fixed maintenance machine list and a production plan which includes the *jobs* that need to be processed, along with their release time and due time. The decision maker then needs to simultaneously determine the job schedule and maintenance schedule. Regarding the job scheduling problem, each job consists of multiple *steps*, and we refer to a *task* as a job in a specific step for the problem description.

doi:10.6342/NTU202400496

The decision maker must assign each task to exactly one machine. Regarding the maintenance scheduling problem, we identify machines that are on the required fixed maintenance machine list as *must-maintenance machines*. The decision maker needs to schedule when to perform maintenance on each of these machines, and that maintenance must be completed by a given due time. The objective of the aforementioned scheduling problem is to minimize the total weighted tardiness of the production schedule. Additionally, in order to avoid excessive gaps between tasks in order to prevent rework. Therefore, these scheduling problems must adhere to the queue time limit for each task.

Let the set of jobs from the given production plan be $I = \{1, 2, 3, ..., n_I\}$, and we need to complete all of the jobs in the planning horizon. Job i has release time R_i, due time D_i, and weighted penalty of tardiness W_i. These jobs need to go through multiple steps $S = \{1, 2, 3, ..., n_S\}$ for production with parallel machines. In this study, some processes are repetitive, and a process contains multiple tasks. These tasks must be processed on the same group of machines. We express a group of machines from a station as $H = \{1, 2, 3, ..., n_H\}$. Moreover, for station h, we denote the set of machines that must be maintained at least once as X_h^P, which is the subset of the set of all machines X.

We use certain decision variables to determine the schedule of jobs and maintenance. We say a job has n_s tasks one for each step, and we denote the task of job i in step s as task (i, s), and all tasks could not be preempted. If a machine processes a task, we could not insert any task to interrupt it. To model the sequence of jobs at the same machine, let the decision variable $y_{ijstm}^{JJ} \in \{0, 1\}$ be 1 when task (i, s) precedes task (j, t) or 0 otherwise. In addition, in order to avoid the over-queue-time situation, the queue time q_{is} between the completion time of task $(i, s-1)$ and task (i, s) could not be longer than

doi:10.6342/NTU202400496

the queue time limit L_{is}. Moreover, if machine m is under maintenance, it cannot run any task at the same time, so we let the decision variable $y_{ism}^{PJ} \in \{0, 1\}$ as 1 if maintenance at machine m precedes task (i, s) or 0 otherwise. It could ensure the maintenance and job do not overlap on the same machine.

In our problem, maintenance takes the amount of time F_h at station h, and the maintenance also cannot be interrupted. The maintenance schedule needs to cover all must-maintenance machines, and the durations of any pair of maintenances at the same station cannot overlap due to the limitation of maintenance capacity. If there are maintenance schedules of the machine m and machine k at the station h, we set the decision variable y_{hmk}^{PP} as 1 if the maintenance at machine m precedes the maintenance at machine k. This variable will be used in constraint to avoid the overlap between maintenance at the same station.

For the end of scheduling, all jobs must complete all steps. If we have the completion time z_{i,n_S} and final due date D_i of job i, tardiness is defined as a maximum of z_{i,n_S} minus D_i or 0. Once z_{i,n_S} is later than D_i, there exists the tardiness for this job, and the tardiness of schedule is the sum of the tardiness of all jobs.

3.2 Industry example

In the semiconductor and panel industry, the joint preventive maintenance and job scheduling problem is a common issue. Take the TFT-LCD process as an example, which consists of three main phases: array, cell, and module. In the array production process of our collaborating company, the array phase is divided into three parts: Thin

doi:10.6342/NTU202400496

Film, Photolithography, and Etching. Specifically, in a collaborating TFT-LCD manufacturer, each job in the array phase consists of 24 steps. These steps are derived from five iterations of the PEP (Photo Engraving Process). Consequently, the job undergoes a repetitive procedure and is processed by the machine at the same station.

The overall process is illustrated in Figure 3.1, we mark the steps of the job as step-station. Take 1-1 as an example, it represents step 1 of the job process by the machine of station 1. If the steps are processed by the different machines from the same station, which means the steps are the same process.

Figure 3.1: The array phase process of TFT-LCD manufacture

Moreover, each station has multiple machines. In Figure 3.2, we depict the flow from step 1 to 3 of the job. For each step, the decision maker will assign the job to one and exactly one machine from the corresponding station, and there is a queue time between steps. The final schedule must prevent queue time from exceeding queue time limit.

doi:10.6342/NTU202400496

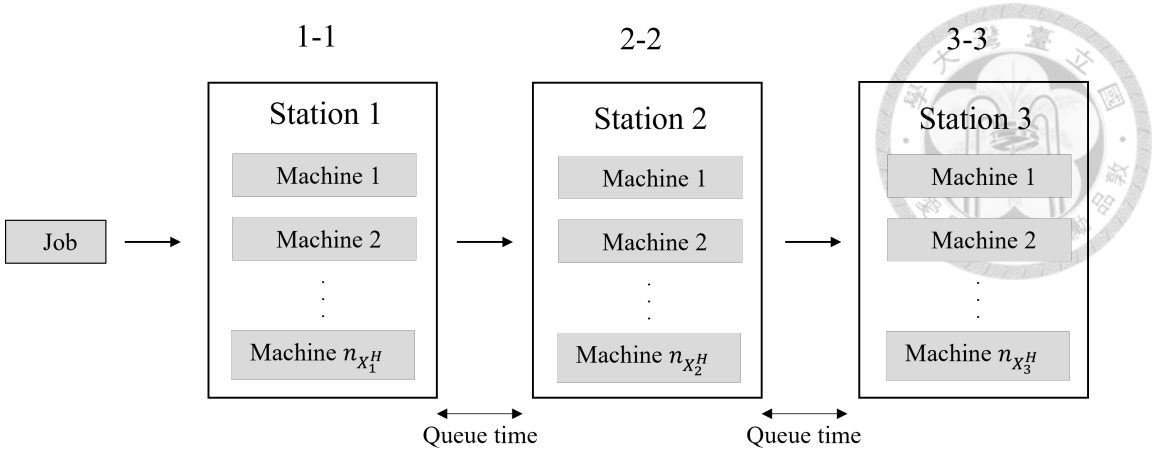


Figure 3.2: The relationship between steps and stations in the array phase

3.3 Model formulation

We use these parameters, variables, and assumptions to formulate a Mixed Integer Program (MIP) that outputs an optimal solution for our joint preventive maintenance and flexible job shop scheduling problem. The objective is to minimize the total weighted tardiness of the production plan.

doi:10.6342/NTU202400496

$$\min \quad \sum_{i \in I} W_i u_i \tag{3.1}$$

$$\text{s.t.} \quad z_{i,n_S} - D_i \leq u_i \quad \forall i \in I \tag{3.2}$$

$$R_i \leq z_{i,1} - \sum_{m \in X_1^S} T_{i,1,m} x_{i,1,m} \quad \forall i \in I \tag{3.3}$$

$$z_{i,s-1} + \sum_{m \in X_s^S} T_{ism} x_{ism} + q_{is} = z_{is} \quad \forall i \in I, s \in \{2, ..., n_S\} \tag{3.4}$$

$$\sum_{m \in X_s^S} x_{ism} = 1 \quad \forall i \in I, s \in S \tag{3.5}$$

$$x_{ism} + x_{jtm} \leq y_{ijstm}^{JJ} + y_{jitsm}^{JJ} + 1 \quad \forall i, j \in I, h \in H, s, t \in S_h, m \in X_h^H \tag{3.6}$$

$$z_{is} + T_{jtm} \leq z_{jt} + M(1 - y_{ijstm}^{JJ}) \quad \forall i, j \in I, h \in H, s, t \in S_h, m \in X_h^H \tag{3.7}$$

$$z_{is} + F_h \leq z_m^P + M y_{ism}^{PJ} + M(1 - x_{ism}) \quad \forall i \in I, h \in H, s \in S_h, m \in X_h^P \tag{3.8}$$

$$z_m^P + T_{ism} \leq z_{is} + M(1 - y_{ism}^{PJ}) + M(1 - x_{ism}) \quad \forall i \in I, h \in H, s \in S_h, m \in X_h^P \tag{3.9}$$

$$z_k^P + F_h \leq z_m^P + M y_{hmk}^{PP} \quad \forall h \in H, m, k \in X_h^P, m \neq k \tag{3.10}$$

$$z_m^P + F_h \leq z_k^P + M(1 - y_{hmk}^{PP}) \quad \forall h \in H, m, k \in X_h^P, m \neq k \tag{3.11}$$

$$q_{is} \leq L_{is} \quad \forall i \in I, s \in \{2, ..., n_S\} \tag{3.12}$$

$$z_m^P \leq B_h \quad \forall h \in H, m \in X_h^P \tag{3.13}$$

$$R_h^P \leq z_m^P - F_h \quad \forall h \in H, m \in X_h^P \tag{3.14}$$

$$y_{ijstm}^{JJ} \in \{0, 1\} \quad \forall i, j \in I, h \in H, s, t \in S_h, m \in X_h^H \tag{3.15}$$

$$y_{ism}^{PJ} \in \{0, 1\} \quad \forall i \in I, h \in H, s \in S_h, m \in X_h^P \tag{3.16}$$

$$y_{hmk}^{PP} \in \{0, 1\} \quad \forall h \in H, m, k \in X_h^P \tag{3.17}$$

$$x_{ism} \in \{0, 1\} \quad \forall i \in I, s \in S, m \in X \tag{3.18}$$

$$q_{is} \geq 0, z_{is} \geq 0 \quad \forall i \in I, s \in S \tag{3.19}$$

$$z_m^P \geq 0 \quad \forall h \in H, m \in X_h^P \tag{3.20}$$

$$u_i \geq 0 \quad \forall i \in I. \tag{3.21}$$

doi:10.6342/NTU202400496

The objective of the model (3.1) is to minimize the total weighted tardiness. If the final completion time of job i is later than the due time, we multiply the weight penalty W_i and tardiness u_i and add it to the objective value. Constraint (3.2) defines the tardiness u_i of job i by the difference between the final completion time z_{i,n_S} and due date D_i.

Constraint (3.3) ensures the job i starts processing the first step after the release time R_i. In addition, in order to limit the machines that can be selected by step s, we define the set X_s^S to be the set of machines that the step s could be assigned. Constraint (3.4) defines the relationship between steps of job i, and also defines the queue time q_{is} of job i in each step s. Constraint (3.5) ensures that each task (i, s), which means job i in step s, be assigned to exactly one machine.

Constraint (3.6) and constraint (3.7) demonstrate the sequence between task (i, s) and task (j, t) in the machine m. Machine m is from the set X_h^H, which includes all machines at station h. Constraint (3.6) ensures that if task (i, s) and task (j, t) are both at machine m, they cannot overlap. If x_{ism} and x_{jtm} are both 1, one of y_{ijstm}^{JJ} and y_{jitsm}^{JJ} will be 1. This then requires constraint (3.7) to limit the two tasks do not overlap. If task (i, s) and task (j, t) are not both on machine m, constraint (3.6) allows y_{ijstm}^{JJ} and y_{jitsm}^{JJ} be 0, then makes constraint (3.7) always satisfy.

Constraint (3.8) and constraint (3.9) ensure task (i, s) and the maintenance at machine m do not conflict. Moreover, the set X_h^P includes all machines in the required fixed maintenance machine list at station h. If $x_{ism} = 1$ and $y_{ism}^{PJ} = 1$, the maintenance must finish before task (i, s) starts or $y_{ism}^{PJ} = 0$ otherwise. If $x_{ism} = 0$, task (i, s) is not at machine m, so these two constraints will relax.

Constraint (3.10) and constraint (3.11) ensure maintenance at machine m and main-

doi:10.6342/NTU202400496

tenance at machine k which are both from station h do not conflict. If $y^{PP}_{hmk} = 1$, maintenance at machine m must be completed before maintenance at machine k or $y^{PP}_{hmk} = 0$ otherwise. Constraint (3.12) let the queue time q_{is} of task (i, s) be shorter than queue time limit L_{is}. As this in effect makes the over-queue-time issue does not happen. Constraint (3.13) and constraint (3.14) ensure that the maintenance is completed within the specified duration.

Constraints (3.15) – (3.18) imply these decision variables are binary variables. Constraints (3.19), constraints (3.20), and constraints (3.21) identify these decision variables are non-negative.

Table 3.1 introduce all the indices and sets, and Table 3.2 introduces all the sets and parameters in the model. Last but not least, Table 3.3 describes all the decision variables for our study.

doi:10.6342/NTU202400496

Table 3.1: List of indices and sets

Indices and Sets	
i, j and n_I	Index of jobs and number of jobs, $i, j = 1, 2, ..., n_I$.
s, t and n_S	Index of steps and number of steps, $s, t = 1, 2, ..., n_S$.
m, k and n_X	Index of machines and number of machines, $m, k = 1, 2, ..., n_X$.
h	Index of stations, $h = 1, 2, ..., n_H$.
I	The set of jobs, $I \in \{1, 2, ..., n_I\}$.
H	The set of stations, $H \in \{1, 2, ..., n_H\}$.
S	The set of steps, $S \in \{1, 2, ..., n_S\}$.
X	The set of machines, $X \in \{1, 2, ..., n_X\}$.
S_h	The set of steps in station h, $S_h \subseteq S$.
X_h^H	The set of machines in station h, $X_h^H \subseteq X$.
X_s^S	The set of machines that could be selected in step s, $X_s^S \subseteq X$.
X_h^P	The set of machines that need to be maintained in station h, $X_h^P \subseteq X$.

doi:10.6342/NTU202400496

Table 3.2: List of parameters

Parameters	
T_{ism}	The processing time of job i in step s at machine m.
L_{is}	The queue time limit of job i in step s.
R_i	The release time of job i.
R_h^P	The maintenance release time of station h.
D_i	The final due time of job i.
W_i	The weight of tardiness of job i.
F_h	The length of each maintenance at station h.
B_h	The limit of each maintenance complete time at station h.
M	A very large positive number.

doi:10.6342/NTU202400496

Table 3.3: List of decision variables

Variables	
z_m^P	The completion time of maintenance at machine m.
z_{is}	The completion time of job i in step s.
q_{is}	The queue time of job i in step s.
y_{ijstm}^{JJ}	A binary variables that verifies whether job i in step s precedes job j in step t at machine m, 1 if job i in step s precedes job j in step t at machine m or 0 otherwise.
y_{ism}^{PJ}	A binary variables that verifies whether maintenance precedes job i in step s at machine m, 1 if maintenance precedes job i in step s at machine m or 0 otherwise.
y_{hmk}^{PP}	A binary variables that verifies whether maintenance at machine m of station h precedes maintenance at machine k of station h, 1 if maintenance at machine m of station h precedes maintenance at machine k of station h or 0 otherwise.
x_{ism}	A binary variables that verifies whether job i in step s is assigned to machine m, 1 if job i in step s is assigned to machine m or 0 otherwise.
u_i	The tardiness of job i.

doi:10.6342/NTU202400496

Chapter 4

Algorithm

Our algorithm consists of two phases. The first phase involves obtaining a feasible solution using a greedy procedure, while the second phase focuses on improving algorithm performance through the genetic algorithm. We propose a greedy procedure that can efficiently generate a good solution within a reasonable timeframe. This procedure comprises three modules: job listing, maintenance scheduling, and job scheduling.

In the job listing module, the greedy procedure generates a sequence of jobs based on specific rules. This sequence determines the order in which the jobs are processed and serves as the basis for generating an initial feasible schedule.

The second module focuses on determining the rules for maintenance scheduling. Preventive maintenance is a crucial task that requires resources and affects machine operation. Therefore, the sequencing of preventive maintenance plays a vital role.

The job scheduling module is the last component of the algorithm. It can be divided into three parts: job assignment, backward scheduling by considering queue time, and

doi:10.6342/NTU202400496

resolution of any issues related to over-queue-time. These three parts aim to minimize the total weighted tardiness while considering the assigned maintenance schedule.

After going through the three modules of the greedy procedure, in order to obtain the best job schedule, our algorithm utilizes the genetic algorithm to optimize the sequence of jobs and improve performance. In this phase, the genetic algorithm continuously generates several initial job sequences and re-executes the job scheduling module with the fixed preventive maintenance schedule. This iterative process eliminates the job schedule with the highest total weighted tardiness and retains the first several schedules with the least total weighted tardiness until the stopping criterion is met.

Upon completion of the two phases of the algorithm, our algorithm proposes the schedule with the least total weighted tardiness from all the considered job schedules.

The whole process is shown in Figure 4.1.

doi:10.6342/NTU202400496

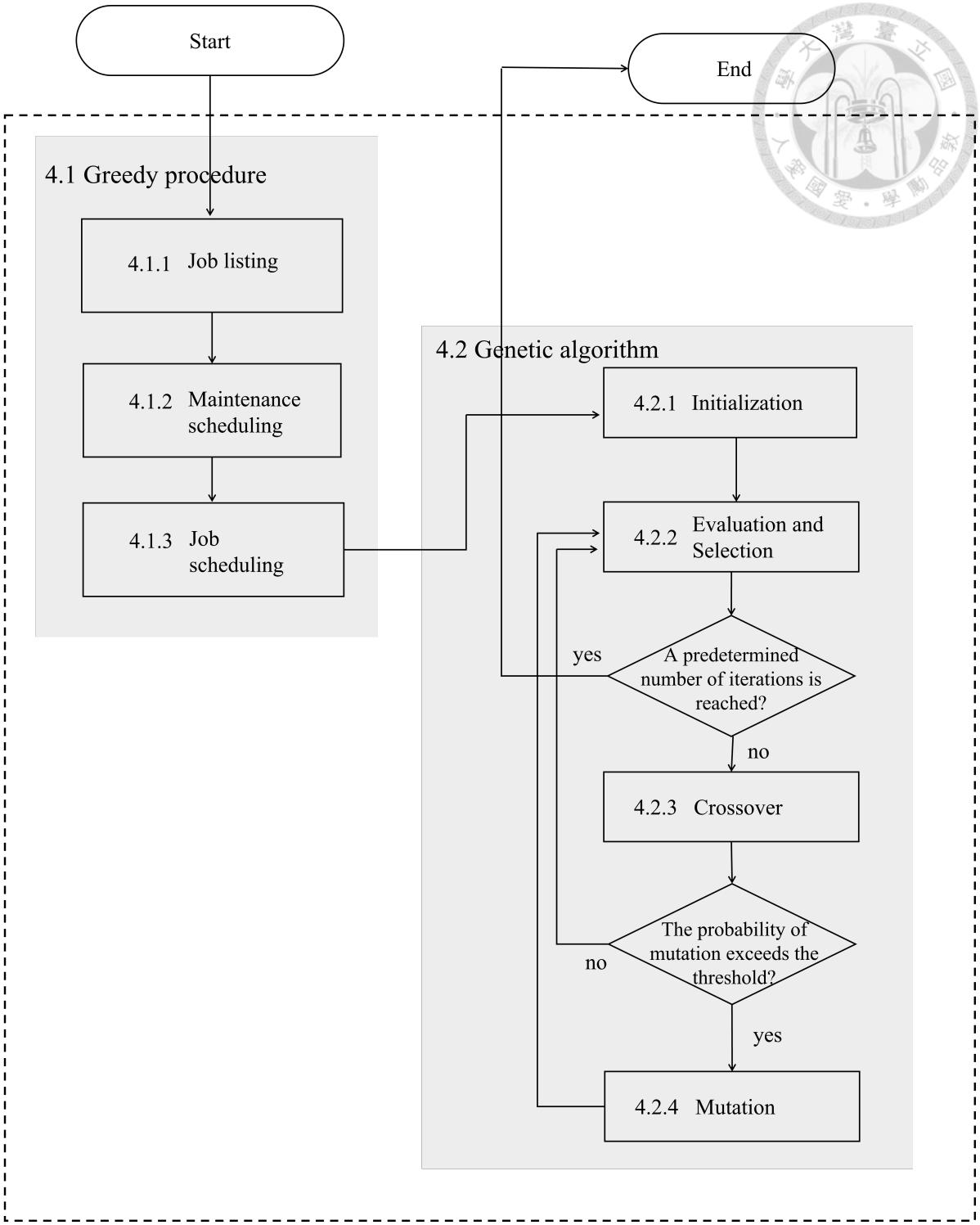


Figure 4.1: The overall process of the algorithms

doi:10.6342/NTU202400496

4.1 Greedy procedure

4.1.1 Job listing

The first module is job listing. Because our problem is to minimize the total weighted tardiness, we need to consider the weight of penalties and tardiness at the same time, so we use "Weighted Earliest Due Date" (WEDD) rule to sort the jobs from smallest to largest (Kanet and Li, 2004). The index of job i under WEDD is

$$\frac{\text{Due time of job } i}{\text{Tardiness penalties of job } i} = \frac{D_i}{W_i}. \tag{4.1}$$

If some jobs have the same score $\frac{D_i}{W_i}$, then the smaller the job ID, the higher the priority. This rule tends to prioritize jobs with earlier due times and larger tardiness penalties. However, this rule does not apply to the job list generated based on WEDD.

4.1.2 Maintenance scheduling

In our study, we need to deal with must-maintenance machines by scheduling machine maintenance at the appropriate time. To minimize the total weight tardiness, our algorithm is designed to arrange the tasks as far ahead as possible. Therefore, we use the logic of scheduling preventive maintenance as late as possible. Additionally, we also consider that the maintenance positions of the same station cannot overlap, the maintenance time of the machine must be later than the maintenance release time, and the maintenance end time must be before the maintenance due time.

To determine the maintenance position for different machines in the same station, we rank the machines based on their average job processing time. The shorter the average

doi:10.6342/NTU202400496

job processing time, the better the condition of the machine, and the more jobs it can complete. In order to allow the machine to handle more jobs, we prioritize scheduling maintenance for machines in better condition. If there are machines with the same average processing time, the machine with the smaller ID is scheduled for maintenance first. Algorithm 1 and Algorithm 2 show the logic of maintenance sorting and maintenance scheduling. Algorithm 1 sorts the machines in descending order based on their average processing time for jobs and passes the sorted list of machines to Algorithm 2. Algorithm 2 schedules maintenance for the machines starting from the end time limits of the maintenance. During this process, Algorithm 2 considers the condition that maintenance on the same station cannot overlap.

Take Figure 4.2 as an example. Only machine 1 and machine 3 of station are in the must-maintenance machines list. Since we want maintenance to start as late as possible, maintenance is scheduled from maintenance due time backward. Moreover, because the average job processing time of machine 3 is shorter than machine 1, maintenance of machine 3 is scheduled earlier than machine 1. In addition, the maintenance positions of these machines do not overlap and the maintenance start time are not earlier than the maintenance release time.

4.1.3 Job scheduling

After the job listing and maintenance scheduling module, we obtain a job list and a maintenance schedule, and the job scheduling module performs job scheduling. The algorithm arranges jobs according to the order specified in the job list. Furthermore, our study imposes a restriction that over-queue-time cannot occur between tasks and cannot

doi:10.6342/NTU202400496

Algorithm 1 Maintenance storing

Input : Job processing time T_{ism} of required maintenance machines X_h^P and the limit of each maintenance complete time B_h at station h

Output: Sorted required maintenance machine list of each station h

foreach *station* h **do**
  foreach *required maintenance machine* X_h^P **do**
    Calculate the average job processing time T_{ism} for the required maintenance machines.
  end
end
foreach *station* h **do**
  Sort required maintenance machines in descending order according to job processing time T_{ism}.
  if *there are some machines sharing the same average job processing time* T_{ism} **then**
    Sort these machines in ascending order according to the ID of the machine.
  end
end

doi:10.6342/NTU202400496

Algorithm 2 Maintenance scheduling

Input : Sorted required maintenance machine list of each station h

Output: Maintenance schedule

```
foreach station h do
    foreach sorted machine list do
        if this is the machine with the longest maintenance time then
            Assign maintenance for the machine, and the expected completion time is B_h.
        else
            Assign maintenance for the machine, and the expected completion time is the
             maintenance start time of the previous machine that requires maintenance.
        end
    end
end
```


doi:10.6342/NTU202400496

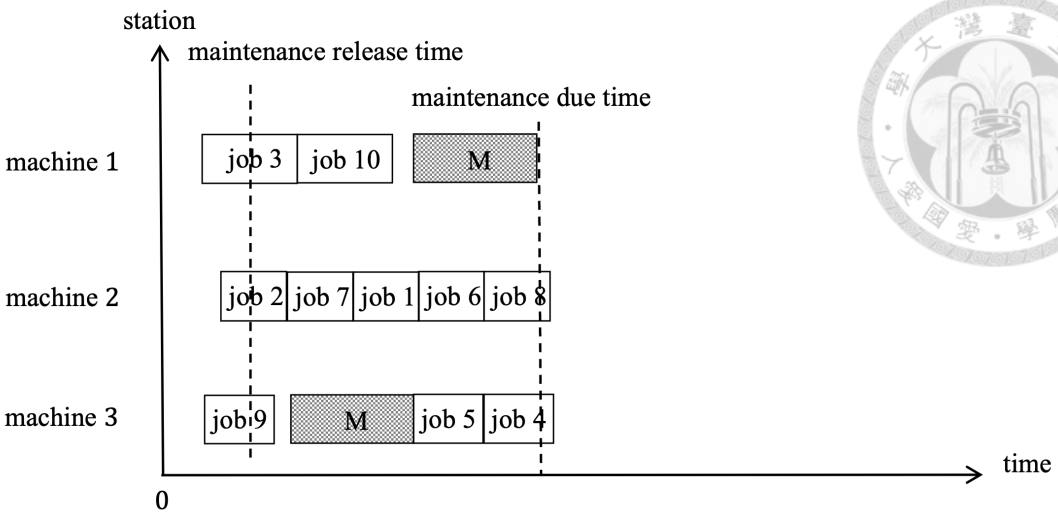


Figure 4.2: A schematic diagram of maintenance scheduling

be violated. To minimize the occurrence of over-queue-time, this algorithm arranges all steps of a job before switching to the next job. This module can be divided into three steps consisting of job assignment, backward scheduling by considering queue time, and solving the over-queue-time issue. The whole process of job scheduling is shown in Figure 4.3.

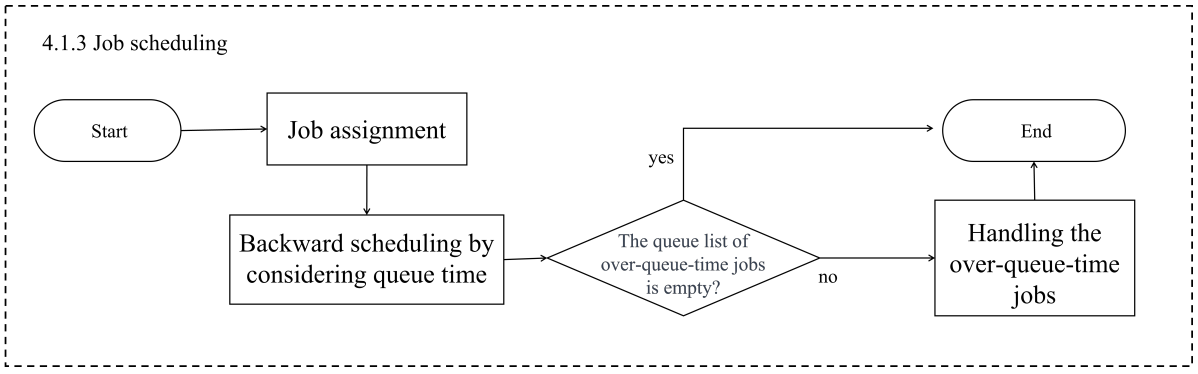


Figure 4.3: The process of the job scheduling

In the job assignment part, we arrange all the steps of the job onto the machine according to the order specified in the job list. When assigning a task, each task will

doi:10.6342/NTU202400496

choose the most suitable machine and be assigned to the earliest available time on the machine. We rank machines for the task based on several conditions and show in the Algorithm 3. First of all, to find the machine that can complete the task earliest, we rank them based on the earliest end time. If there are multiple machines with the same result, we sort them according to the start time of the task if the task is assigned to the machine, from earliest to latest. If there are still multiple machines with the same result, we then sort them based on the average job processing time of the task. If there are machines with the same score after applying the three criteria mentioned above, the algorithm will choose the machine with the smaller ID to assign the task. Moreover, if the step of the task being processed is the first step, the earliest available time will be searched after the job release time. On the other hand, if it is any other step, the search will be conducted after the end time of the previous step. Algorithm 4 shows the logic of job assignment.

After assigning all the tasks, we use a backward check method to inspect if there are any occurrences of over-queue-time for the current task. Because the bottleneck station has the highest workload, we will try to avoid any idle time at the bottleneck station. Therefore, when checking for the occurrence of over-queue-time issue, we start from the bottleneck station and check the preceding stations. If over-queue-time occurs, we check if moving the previous step backward can resolve the over-queue-time issue. If it is possible, we directly move the step backward and continue checking the preceding steps until all steps are checked. If it is not possible, we remove all the steps of this job from the schedule and put them in the pending list for over-queue-time issue, awaiting processing in the next phase. Algorithm 5 shows the logic of backward scheduling by considering queue time.

doi:10.6342/NTU202400496

Algorithm 3 Machine ranking

Input : Current status of each machine and processing time of the job on each machine

Output: Sorted machine list

```
foreach the sorted job list do
    Sort machines in ascending order according to job completion time.
    if there are some machines sharing the same job completion time then
        Sort these machines in ascending order according to job start time.
        if there are some machines sharing the same job start time then
            Sort these machines in ascending order according to the average job processing
             time.
            if there are some machines sharing the same average job processing time then
                Sort these machines in ascending order according to the ID of the machine.
            end
        end
    end
end
```


doi:10.6342/NTU202400496

Algorithm 4 Job assignment

Input : The sorted job list from the job listing part and sorted machine list

Output: Original schedule including job and maintenance

```
1 foreach i in the sorted job list do
2     foreach s in S do
3         Select the most suitable machine for the task and assign the task.
          if this is first step then
4             The start time of the task must be later than the release time R_i.
5         else
6             The start time of the task must be later than the completion time of the
              previous task.
7         end
8     end
9 end
```


doi:10.6342/NTU202400496

Algorithm 5 Backward scheduling by considering queue time

Input : Original schedule including job and maintenance, the sorted job list, bottleneck station ID list, and the steps that the bottleneck station needs to check

Output: Schedule including job and maintenance after backward check and the pending list

```
1  foreach i in the sorted job list do
2      foreach h in bottleneck station ID list do
3          foreach the steps that the bottleneck station needs to check in descending order
            do
4              if there is an over queue time situation then
5                  Try to pull the previous step back to a position that can solve the over-
                    queue-time. if there is no space that can solve the over queue time then
6                      Take all the steps of the job out of the schedule and put them in the
                        pending list.
7                  else
8                      Pull the previous step back to a position to solve the over-queue-time.
9                  end
10             end
11         end
12     end
13 end
```


doi:10.6342/NTU202400496

Figure 4.4 shows a scenario of a single machine where over-queue-time occurs. In the y-axis of the figure, take h1-m1 as an example, h1 stands for station 1, and m1 stands for machine 1. In this case, station 3 is the bottleneck station, so the algorithm starts from station 3 and checks if there is over-queue-time during steps. For job 3, because the queue time between step 2 and step 3 is longer than the queue time limit of step 3, the algorithm takes into account the space between the current end time of step 2 and the start time of step 3 and tries to move step 2 backward to a position where the over-queue-time issue can be resolved. If the over-queue-time issue cannot be solved, all steps of job 3 will be rescheduled.

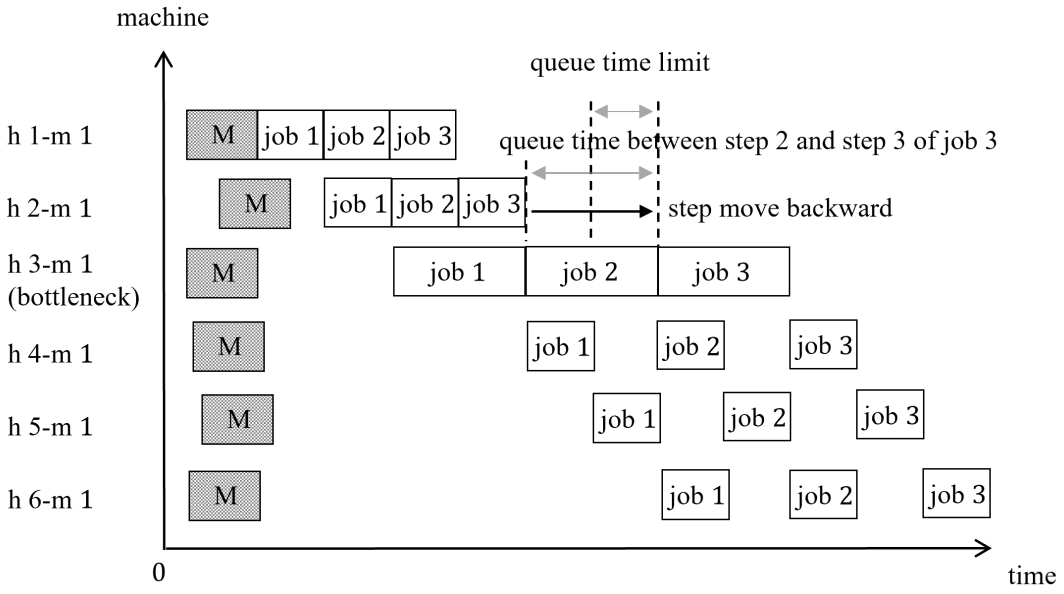


Figure 4.4: A schematic diagram of the backward scheduling by considering queue time before adjustment

Figure 4.5 shows the result of successfully solving the over-queue-time issue by moving step 2 backward.

In the last step of the job scheduling module, the algorithm schedules all the jobs

doi:10.6342/NTU202400496

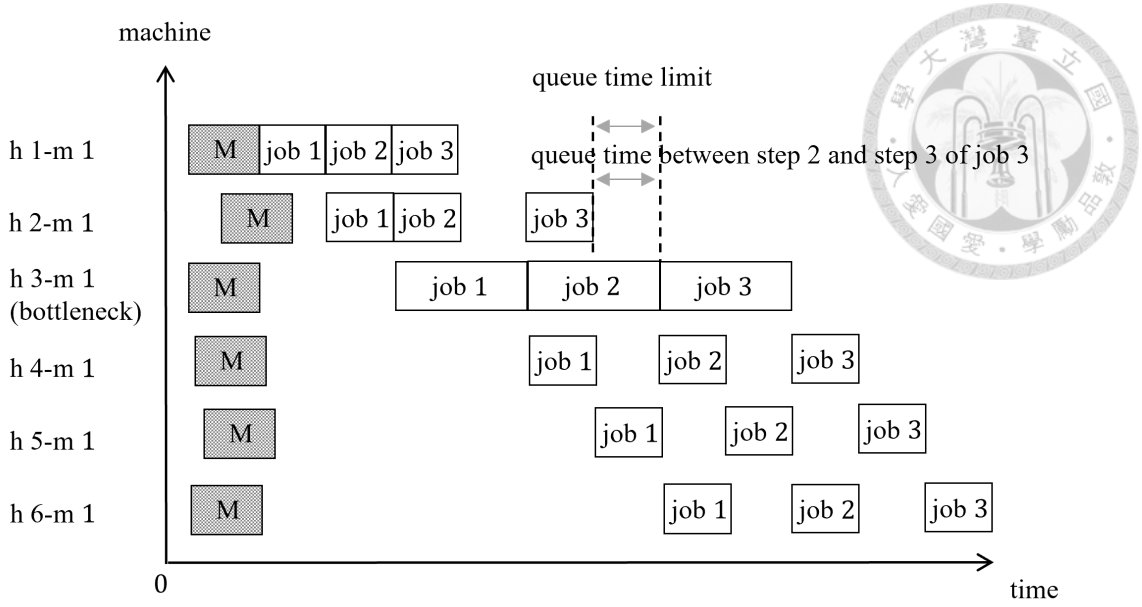


Figure 4.5: A schematic diagram of the backward scheduling by considering queue time after adjustment

that have been placed in the pending list due to the over-queue-time issue. It sorts these jobs using the same logic as the general jobs, but in this case, it only considers the job end time on the machine. The algorithm takes into account all the available time slots where the end time of the previous step of the same job does not overlap with the start time of the next step and is long enough to schedule the task. During the task scheduling process, the algorithm also checks for instances of exceeding the queue time limit. If the over-queue-time issue occurs, the algorithm will switch to assigning a different machine until the queue time issue is resolved. After the above steps are completed, a reasonable and good enough scheduling result is obtained. Algorithm 6 shows the logic of Handling the over-queue-time jobs.

doi:10.6342/NTU202400496

Algorithm 6 Handling the over-queue-time jobs

Input : Original schedule including job and maintenance, the sorted job list, bottleneck station ID list, and the steps that the bottleneck station needs to check

Output: Feasible solution

```
1  foreach i in the pending list do
2      foreach s in S do
3          Get the sorted machine list.
           (The machines are sorted through the machine ranking algorithm, but only the
           space at the end of the machines is considered here.)
           while until the task has been scheduled do
4              Select the most suitable machine for the task and assign the task.
               if there is an over queue time situation then
5                  if there are still machines suitable for assignment then
6                      Change the machine to be assigned the task.
7                  else
8                      The start time of the task must be later than the release time R_i.
9                  end
10             else
11                 Assign the task.
12             end
13         end
14     end
15 end
```


doi:10.6342/NTU202400496

4.2 Genetic algorithm

To improve the algorithm's performance, we use the genetic algorithm to generate and explore various job sequences. Our genetic algorithm consists of five parts: initialization, evaluation, selection, crossover, and mutation.

4.2.1 Job lists initializating

For the genetic algorithm, we need to generate multiple initial job lists, which will be placed in the population pool and considered in the next steps. We set the number of initial job lists to 20. These job lists are generated using several rules, including the WEDD rule, Earliest Due Date rule (EDD), First Come First Serve (FCFS) rule and random generation. The first rule, WEDD, is the same as described earlier. The second rule, EDD, sorts jobs based on their due times, with the earlier due time jobs being considered first. The difference between the WEDD rule and the EDD rule is that the EDD rule ignores the weight of jobs. The third rule, FCFS, prioritizes jobs based on their release times, with the job having the earlier release time being placed first. These three rules give priority to the job with the smaller ID if the jobs have the same score according to the listing rule. Except for the lists generated by the above three rules, the remaining lists are generated randomly.

4.2.2 Schedule evaluating and parents selecting

After generating the initial population pool, we use these lists in our job scheduling module to evaluate the total weighted tardiness of the resulting schedules. These evaluation

doi:10.6342/NTU202400496

scores will be used as criteria for the selection phase. In the selection phase, we need to select two lists as parents for the crossover phase. The selection criteria are calculated by determining the proportion of the total weighted tardiness of schedules generated by each list, and the list with a higher proportion will have a higher chance of being selected. Since we aim to minimize the total weighted tardiness, the proportion will be higher when the total weighted tardiness is lower.

4.2.3 Crossovering

In the crossover part, we use the job lists which are selected in the selection phase to be the basic sequences for crossover. These lists are referred to as parents because we use them to generate new job lists which are called offspring. We refer to the parent with the smaller total weighted tardiness as the first list and the other as the second list. The method of crossover is to specify a ratio, which will be used as the ratio to calculate the position of the cut from the first job in the first list. We split the first list from the specified position and fix the sequences of these two sub-sequences. To the first half sub-list of the first list, we add the missing jobs from the end of the list with the order of these jobs in the second list and then obtain the first offspring. Moreover, for the second half sub-list, we insert the missing jobs from the beginning of the sub-list and also follow the order of these jobs in the second list, which is the second offspring. Take Figure 4.6 as an example. In this case, there are 10 jobs that need to be done, and the cut ratio is 0.6. Because the cut ratio is calculated from the first job of the first list which has lower total weighted tardiness than another selected list, the cut position is 6.

Figure 4.7 shows that the first offspring keeps the first half sub-list of the first list and

doi:10.6342/NTU202400496

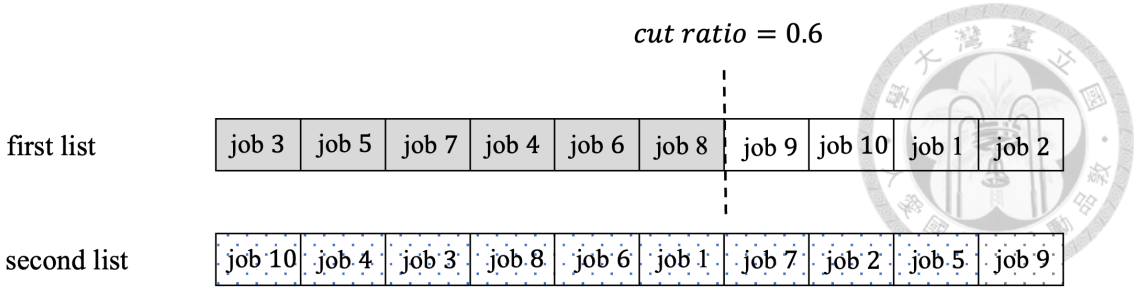


Figure 4.6: A schematic diagram of cut position

adds the rest of the jobs from the end of the first offspring with the order of these jobs in the second list. For example, the missing jobs of the first half sub-list are job 1, job 2, job 9, and job 10, and the order of these jobs in the second list is job 10, job 1, job 2 then job 9. These jobs are added to the first offspring from the back in this order. On the other hand, the second offspring keeps the second half sub-list of the first list to be tail and inserts the missing jobs with the order which is the same as the order of these jobs appear in the second list. These two offspring will be the output at the end of each crossover part.

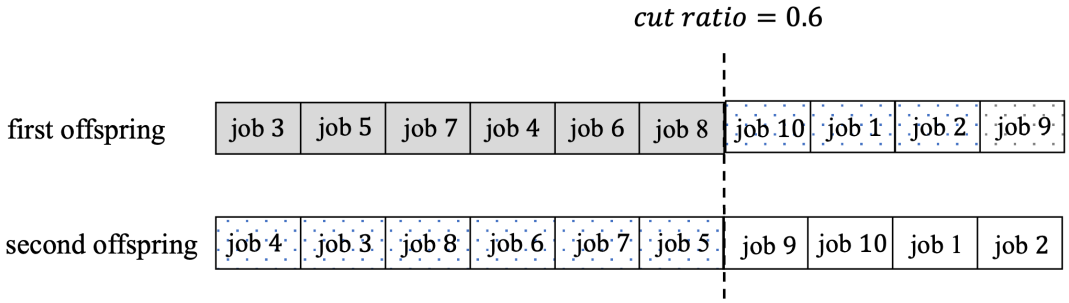


Figure 4.7: A schematic diagram of crossover result

doi:10.6342/NTU202400496

4.2.4 Mutating

The last step in the genetic algorithm is mutation. In the mutation part, we set a probability that each offspring obtained through crossover will occur mutation. The method of mutation is to select two jobs from the offspring and exchange them. As shown in Figure 4.8, after the mutation, the algorithm randomly selects job 6 and job 10 in the offspring to exchange, and the mutated offspring will replace the offspring without mutation to enter the evaluation stage.

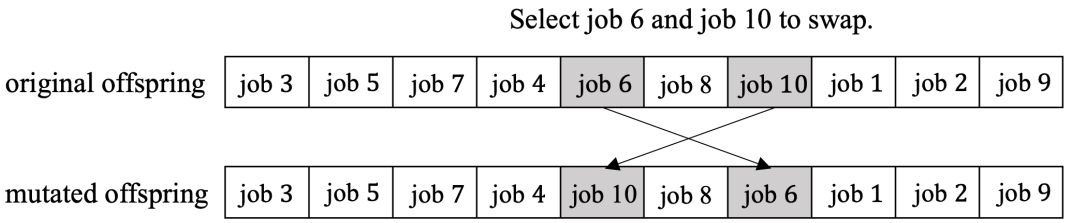


Figure 4.8: A schematic diagram of mutation

The genetic algorithm keeps repeating the four steps of valuation, selection, crossover and mutation until it encounters a stopping criterion and gives a job list which is used as the order of job scheduling with minimum total weighted tardiness in the considered job lists.

doi:10.6342/NTU202400496

Chapter 5

Performance Evaluation

5.1 Experiment design

In the experiment of our study, we set the number of jobs as $n_I = 10$ and the number of steps for each job as $n_s = 6$. Based on this setup, the experiment consists of four major environments: multiple machines with re-entrant, multiple machines with no re-entrant, single machine with re-entrant, and single machine with no re-entrant. Regarding the number of machines n_X, if the environment is multi-machine, we set the number of machines in each station as 3; otherwise, it is set to 1. For determining the number of stations n_H, considering that we have 6 steps in the experiment when there is no re-entrant environment, the number of stations is $n_H = 6$, and the bottleneck station is station 3; otherwise, it is set to $n_H = 3$, and the bottleneck station is station 2. Table 5.1 presents the settings of the environments, including the number of stations, the bottleneck station ID, and the number of machines at each station.

doi:10.6342/NTU202400496

Table 5.1: The setting of environments

Environment name	number of station	bottleneck station ID	number of machines of station
multiple machines with re-entrant	3	2	3
multiple machines with no re-entrant	6	3	3
single machine with re-entrant	3	2	1
single machine with no re-entrant	6	3	1

In other numerical settings, we utilize the concept of a multiplier. Take the example of job processing time. In our experiment, if the processing time for scenario A follows a uniform distribution between 5 and 10 and is three times that of scenario B, then the processing time for scenario B will follow a uniform distribution between 20 and 25. To obtain the number 20, we multiply the average of 5 and 10 by three and subtract 2.5. Similarly, to obtain the number 25, we multiply the average of 5 and 10 by three and add 2.5.

For each task on each machine, the processing time T_{ism} is uniformly distributed between 5 and 10. All machines on the same station have the same processing time for the task. It is important to note that the job processing time on the bottleneck station will be three times that of the general station. This means that the job processing time on the bottleneck station is uniformly distributed between 20 and 25. The queue time limit of each task L_{is} is 14 times the job processing time. This implies that the queue time limit

doi:10.6342/NTU202400496

of each task is uniformly distributed between 102.5 and 107.5, and it is rounded. The 14 times job processing time is a reference to the data of TFT-LCD panel production. The maintenance length F_h for each machine is uniformly distributed between 5 and 10. In the case of a multi-machine scenario, the experiment will randomly select two machines out of three for maintenance; otherwise, it will maintain the only machine available. For each job, the weight of tardiness W_i is uniformly distributed between 1 and 10. The due time is related to the job processing time and we set the average proportion of jobs that can meet the due time P to control the urgency of orders. Here is set up 70 percent of jobs are not tardy. The formula for calculating due time is as follows

$$\begin{aligned}
&[\text{ Average processing time of general station} \\
&\times (\text{ Number of steps } - \text{ Number of steps on bottleneck station }) \\
&+ \text{ Average processing time of bottleneck station} \\
&\times (\text{ Number of steps on bottleneck station } - 1)] \\
&+ \text{ Number of steps on bottleneck station } \times \text{ Number of jobs } \times P
\end{aligned} \tag{5.1}$$

Figure 5.1 displays the due time setting of the single-machine scenario. In this example, the number of jobs n_I is 3, and the number of steps n_s is 6. In addition, we set the percentage P as 70 so that 70 percent of jobs are not tardy, which means that in our case, 2 jobs are completed before the due time D_i.

In order to verify the performance of our algorithm, we employ five factors to conduct an analysis of the performance under diverse conditions. In order to compare the effects of different factors, we consider the above-mentioned settings as the base case and design other 10 scenarios for four environments by adjusting these five factors, which include processing time T_{ism} of bottleneck station, queue time limit L_{is}, the average proportion

doi:10.6342/NTU202400496

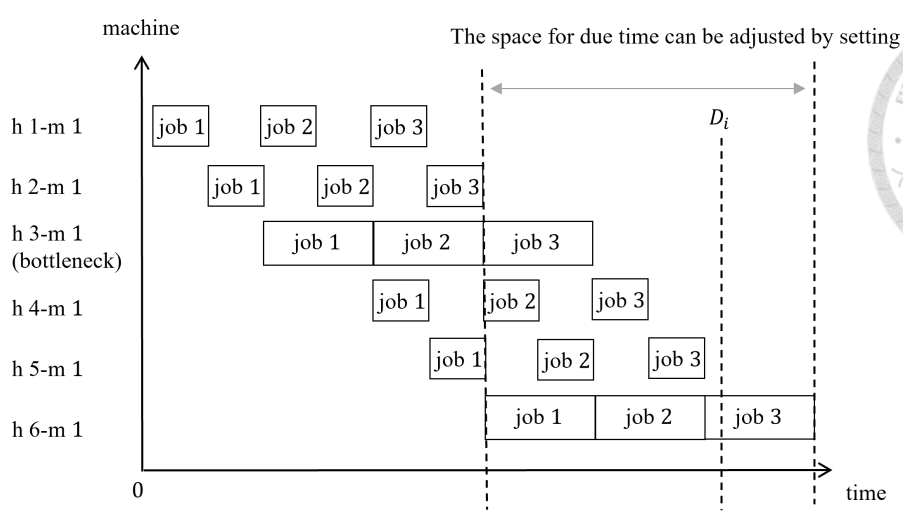


Figure 5.1: A schematic diagram of due time setting

of jobs that can meet the due time P, the length of maintenance F_h, and the weighted penalty of tardiness W_i.

Table 5.2: The setting of scenarios

Scenario	Scenario name	T_{ism} of bottleneck station	L_{is}	P	F_h	Wi
1	base case	$U[20, 25]$	$U[102.5, 107.5]$	70	$U[5, 10]$	$U[1, 10]$
2	no bottleneck	$U[5, 10]$	$U[102.5, 107.55]$	70	$U[5, 10]$	$U[1, 10]$
3	significant bottleneck	$U[35, 40]$	$U[102.5, 107.5]$	70	$U[5, 10]$	$U[1, 10]$
4	tight queue time limit	$U[20, 25]$	$U[50, 55]$	70	$U[5, 10]$	$U[1, 10]$
5	loose queue time limit	$U[20, 25]$	99999	70	$U[5, 10]$	$U[1, 10]$
6	tight due time	$U[20, 25]$	$U[102.5, 107.5]$	50	$U[5, 10]$	$U[1, 10]$
7	loose due time	$U[20, 25]$	$U[102.5, 107.5]$	90	$U[5, 10]$	$U[1, 10]$
8	short maintenance length	$U[20, 25]$	$U[102.5, 107.5]$	70	0	$U[1, 10]$
9	long maintenance length	$U[20, 25]$	$U[102.5, 107.5]$	70	$U[72.5, 77.5]$	$U[1, 10]$
10	no weight	$U[20, 25]$	$U[102.5, 107.5]$	70	$U[5, 10]$	1
11	obvious weight discrepancy	$U[20, 25]$	$U[102.5, 107.5]$	70	$U[5, 10]$	1 or 10

doi:10.6342/NTU202400496

The significance of the bottleneck is a crucial factor in our algorithm, as the backward scheduling by considering queue time phase in the greedy procedure adjusts the schedule based on the condition of the bottleneck station. Therefore, we design two scenarios to compare the experiment with the base case: one with a significant bottleneck station and one without a bottleneck station. We adjust the processing time to determine the obviousness of the bottleneck station. Additionally, we designate the intermediate process station as the bottleneck station to better align with the TFT-LCD process. Therefore, in the case of re-entrant, we specify station 2 as the bottleneck station, and in the absence of re-entrant, we designate station 3 as the bottleneck station. In the base case, the processing time at the bottleneck station is three times that of the original station, following a uniform distribution between 20 and 25. However, in a significant bottleneck scenario, the processing time at the bottleneck station becomes five times that of the original station, following a uniform distribution between 35 and 40. On the other hand, if there is no bottleneck, the processing time at the bottleneck station is the same as that of the original station.

The second factor is the queue time limit of task L_{is}. In the base case, we set the queue time limit L_{is} to 14 times the processing time of the original station according to the TFT-LCD process, which follows a uniform distribution between 102.5 and 107.5. Additionally, we establish both a loose queue time limit and a tight queue time limit. In the case of the loose queue time limit scenario, we set the queue time limit to a very large number, 99999, indicating no queue time limit. For the tight queue time limit scenario, we set the queue time limit L_{is} to 7 times the processing time of the original station, resulting in a uniform distribution between 50 and 55.

doi:10.6342/NTU202400496

The next factor is to set the degree of urgency of the order by adjusting the average proportion of jobs that can meet the due time P. The formula is the same in all cases, except that in the base case, the average proportion of jobs that can meet the due time is set to 70. This means that 70 percent of the jobs are expected to be completed by the due time. In the tight due time scenario, the average proportion of jobs that can meet the due time P will be set to 50. This means that for more urgent orders, only 50 percent of the jobs are expected to be completed by the due time. In the loose due time scenario, the average proportion of jobs that can meet the due time P will be set to 90. This means that orders with more time allow for a higher average proportion of jobs, namely 90 percent, to be completed by the due time.

The factor of the length of maintenance F_h is also an important crucial factor for scheduling. We use this factor to observe the performance of our algorithm for different maintenance lengths. In the base case, we set the length of maintenance F_h as long as the processing time. In the case of the short maintenance length scenario, we set the length of maintenance to 0 to simulate the scenario of no maintenance. For the long maintenance length scenario, we set the length of maintenance to 10 times the processing time of the original station, resulting in a uniform distribution between 72.5 and 77.5.

The last factor is the weighted penalty of tardiness W_i. We observe the impact of the discrepancy of weight between jobs on the performance of the algorithm. In the base case, the weighted penalty of tardiness W_i is following a uniform distribution between 1 and 10. In the no weight scenario, we set the W_i to 1. It means that each job has the same penalty weight for tardiness. For the obvious weight discrepancy scenario, the weighted penalty of tardiness is 0 or 1, which means that some job tardiness will cause

doi:10.6342/NTU202400496

relatively large losses.

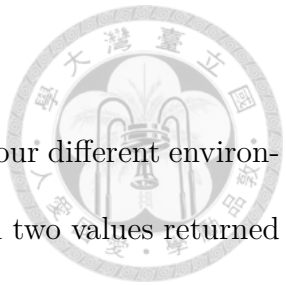
By adjusting the above factors, in addition to the base case, there are 10 scenarios. In each scenario, all settings remain the same as the base case, except for one specific factor that is being modified.

Moreover, the genetic algorithm experiment has the following parameters set. First, we set the number of job sequences that the genetic algorithm needs to generate at the beginning to 20. Then, the probability of mutations in the offspring of the genetic algorithm is 1 percent. Finally, for each scenario, the genetic algorithm will iterate 10,000 times. However, if the best total weight tardiness remains the same for 2,000 consecutive iterations, which accounts for one fifth of the 10,000 runs, the genetic algorithm will stop.

The experiments are conducted with a PC of Intel(R) CoreTM i7-8700 CPU @ 3.20 GHz $\times$ 12 and 15.5 Gibibyte memory. The heuristic algorithms were implemented by Python 3.8.5. For the experiment of the MIP model, we use Gurobi Optimizer 10.0.0 as our solver to solve by a Python program and set 2,700 seconds as the time limit; therefore, the model can solve the problem in a reasonable time. After the calculation stops, we will get the theoretical lower bound and the objective value of the best feasible solution at present. If the best solution is found before the time limit is reached, the solver will stop, and the theoretical lower bound and the objective value of the best feasible solution at that time will be the same. These values are then used to be compared with the values obtained by our proposed algorithms.

doi:10.6342/NTU202400496

5.2 Solution performance



In this section, we evaluate the effectiveness of the algorithm under four different environments. We assess the algorithm's performance by comparing it with two values returned by the solver: the theoretical lower bound and the objective value of the best feasible solution obtained before reaching the time limit, while also comparing their runtimes.

Each scenario contains 20 instances. We calculate the average objective value from these 20 instances and then determine the gap between our algorithm and the MIP model. The formula for the optimality gap is $\frac{z-z^{PB}}{z^{PB}}$ or $\frac{z-z^{TB}}{z^{TB}}$, where z^{PB} is the primal lower bound when the MIP reaches the time limit and z^{TB} is the theoretical lower bound, and z is the objective value calculated using our algorithm. The tables below present the average optimality gap across all scenarios and the computation time for four environments.

5.2.1 Single-machine environment

In a single-machine environment, except in scenarios with a significant bottleneck in the re-entrant environment, the MIP model can find the optimal solution within the time limit. Table 5.3 contains the averages and standard deviations of the two optimality gaps of our greedy approach of the 11 scenarios under the single-machine environment. Moreover, it shows a comparison between the re-entrant and no re-entrant environments, indicating that the performance of the greedy procedure without re-entrant is better in all scenarios. They achieve an average optimality gap of 0.0819 with a standard deviation of 0.0619, when there is no re-entrant. Among these scenarios of no re-entrant environment, the performance is particularly strong when there are the significant bottleneck scenario

doi:10.6342/NTU202400496

and long maintenance length scenario.

Table 5.3: The average optimality gap in the single-machine environment is compared to the greedy procedure

Scenario name	re-entrant				no re-entrant			
	$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$		$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$	
	mean	SD	mean	SD	mean	SD	mean	SD
base case	0.8362	0.1975	0.8362	0.1975	0.0756	0.0484	0.0756	0.0484
no bottleneck	0.3127	0.1225	0.3127	0.1225	0.1181	0.0919	0.1181	0.0919
significant bottleneck	0.4382	0.0869	0.5324	0.1222	0.0393	0.0312	0.0393	0.0312
tight queue time limit	0.8788	0.2069	0.8788	0.2069	0.0892	0.0688	0.0892	0.0688
loose queue time limit	0.8587	0.1653	0.8587	0.1653	0.0728	0.0546	0.0728	0.0546
tight due time	0.6153	0.1256	0.6153	0.1256	0.0559	0.0353	0.0559	0.0353
loose due time	1.0844	0.1838	1.1798	0.5712	0.1081	0.1088	0.1081	0.1088
short maintenance length	0.8740	0.1705	0.8740	0.1705	0.1121	0.0632	0.1121	0.0632
long maintenance length	0.5001	0.1041	0.5001	0.1041	0.0378	0.0187	0.0378	0.0187
no weight	0.6629	0.1143	0.9039	0.1244	0.0791	0.0386	0.0791	0.0386
obvious weight discrepancy	1.1630	0.4211	1.2348	0.4655	0.1132	0.1211	0.1132	0.1211
Average	0.7477	0.1726	0.7933	0.216	0.0819	0.0619	0.0819	0.0619

Table 5.4 contains the optimality gap which is compared to the genetic algorithm in a single-machine environment. It shows that the performance of the genetic algorithm in a single-machine environment is nearly the same as that of the greedy procedure. This

doi:10.6342/NTU202400496

suggests that the WEDD rule may be an appropriate job processing order-generated rule for this problem environment.

Table 5.4: The average optimality gap in the single-machine environment is compared to the genetic algorithm

Scenario name	re-entrant				no re-entrant			
	$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$		$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$	
	mean	SD	mean	SD	mean	SD	mean	SD
base case	0.8362	0.1975	0.8362	0.1975	0.0756	0.0484	0.0756	0.0484
no bottleneck	0.3127	0.1225	0.3127	0.1225	0.1181	0.0919	0.1181	0.0919
significant bottleneck	0.4381	0.0869	0.5324	0.1222	0.0393	0.0312	0.0393	0.0312
tight queue time limit	0.8788	0.2069	0.8788	0.2069	0.0892	0.0688	0.0892	0.0688
loose queue time limit	0.8587	0.1653	0.8587	0.1653	0.0728	0.0546	0.0728	0.0546
tight due time	0.6153	0.1256	0.6153	0.1256	0.0559	0.0353	0.0559	0.0353
loose due time	1.0844	0.1838	1.1798	0.5712	0.1081	0.1088	0.1081	0.1088
short maintenance length	0.8740	0.1705	0.8740	0.1705	0.1121	0.0632	0.1121	0.0632
long maintenance length	0.5001	0.1041	0.5001	0.1041	0.0378	0.0187	0.0378	0.0187
no weight	0.6629	0.1143	0.9039	0.1244	0.0791	0.0386	0.0791	0.0386
obvious weight discrepancy	1.1630	0.4211	1.2348	0.4655	0.1132	0.1211	0.1132	0.1211
Average	0.7477	0.1726	0.7933	0.216	0.0819	0.0619	0.0819	0.0619

Table 5.5 contains the computation time of the three methods of all of the scenarios in the single-machine environment. From Table 5.5, although we can observe that the

doi:10.6342/NTU202400496

MIP model can obtain the optimal solution within a reasonable time, the execution time of greedy procedure is significantly less than that required by the MIP model, especially in significant bottleneck scenario and no weight scenario with re-entrant.

Table 5.5: The computation time (sec) in the single-machine environment

Scenario name	re-entrant			no re-entrant		
	Greedy	Genetic	MIP	Greedy	Genetic	MIP
base case	0.0024	7.2729	93.3028	0.0015	5.2427	26.8282
no bottleneck	0.0019	6.7279	342.3051	0.0016	5.1074	75.2300
significant bottleneck	0.0026	7.4414	1669.8324	0.0025	5.6397	46.5264
tight queue time limit	0.0025	7.3787	75.6792	0.0017	5.6457	26.6735
loose queue time limit	0.0032	6.8143	84.8141	0.0016	5.0613	34.6784
tight due time	0.0019	7.3976	38.0459	0.0022	5.2567	3.4198
loose due time	0.0021	7.3594	297.9902	0.0014	5.2564	94.0999
short maintenance length	0.0021	7.3238	113.3572	0.0016	5.2762	53.7737
long maintenance length	0.0019	7.4082	10.4253	0.0016	5.3486	1.2651
no weight	0.0026	7.3406	2700.0261	0.0017	5.1745	314.148
obvious weight discrepancy	0.0019	7.3838	1469.1733	0.0016	5.1851	22.9144
Average	0.0023	7.259	626.8138	0.0017	5.2904	63.5961

doi:10.6342/NTU202400496

5.2.2 Multiple-machine environment

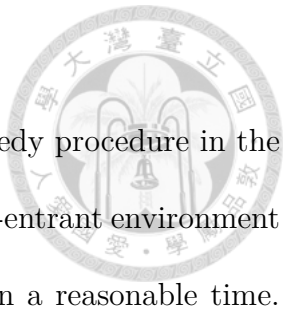


Table 5.6 contains the optimality gap which is compared to the greedy procedure in the multi-machine environment. It shows that in a multi-machine and re-entrant environment situatin, the MIP model fails to obtain the optimal solution within a reasonable time. Although there is a significant optimality gap between the algorithm and the theoretical lower bound, compared to the objective value obtained by the MIP model with an average time of 2700.0354 seconds, an average optimality gap of 0.2353 can be achieved. Based on the optimality gap compared to the objective value obtained by the MIP model with the time limit, among all scenarios, the one with no bottleneck and the one with tight due time demonstrates better performance.

Moreover, in a multi-machine and no re-entrant environment, our algorithm performs better than in the re-entrant environment. Although the MIP model did not obtain the optimal solution for all instances within a reasonable time frame, the algorithm calculated an average optimality gap with the solutions obtained under the time constraint of only 0.0346 with a standard deviation of 0.0269. Based on the optimality gap compared to the objective value obtained by the MIP model with the time limit, all scenarios, the scenarios with long maintenance length and with tight due demonstrate the best performance.

Table 5.7 includes the optimality gap, which is being compared to the genetic algorithm within a multi-machine in the multi-machine environment. Furthermore, it shows the average optimality gap in the multiple-machine environment compared to the objective value of the genetic algorithm. In a multi-machine environment, just like in a single-machine environment, the genetic algorithm performs similarly to the greedy pro-

doi:10.6342/NTU202400496

Table 5.6: The average optimality gap in the multiple-machine environment is compared to the greedy procedure

Scenario name	re-entrant				no re-entrant			
	$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$		$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$	
	mean	SD	mean	SD	mean	SD	mean	SD
base case	0.1992	0.1305	2.9241	1.3093	0.0272	0.0253	3.2452	7.0354
no bottleneck	0.0941	0.0328	0.6766	0.3231	0.0363	0.0287	0.2144	0.1751
significant bottleneck	0.2030	0.0432	26.4989	77.5617	0.0366	0.0266	0.2687	0.4498
tight queue time limit	0.2417	0.1105	2.8983	1.1980	0.0314	0.0191	2.3886	1.9634
loose queue time limit	0.1996	0.1099	2.1221	0.8711	0.0275	0.0242	2.4312	3.6690
tight due time	0.1632	0.0864	1.2518	0.4374	0.0236	0.0195	0.6751	0.3618
loose due time	0.3898	0.1994	28.9644	28.2586	0.0485	0.0468	4.3113	14.2811
short maintenance length	0.3840	0.1095	2.8477	1.2927	0.0545	0.0394	0.0823	0.1242
long maintenance length	0.2455	0.0551	3.9953	1.4999	0.0205	0.0112	0.2374	0.3594
no weight	0.1988	0.0873	4.6632	2.6473	0.0286	0.0226	8.0112	5.3996
obvious weight discrepancy	0.2691	0.1512	1.9018	1.401	0.0460	0.0330	0.9454	0.9994
Average	0.2353	0.1014	7.1586	10.6182	0.0346	0.0269	2.0737	3.1653

cedure. It is possible that the job order of processing generated by the WEDD rule is most beneficial for addressing this type of problem.

In a multi-machine environment, the execution time of the algorithm has a significant advantage compared to the execution time of the MIP model. Table 5.8 presents the time taken for computation using three different methods across various scenarios in a multi-machine setup. It shows that, under the re-entrant environment, the average

doi:10.6342/NTU202400496

Table 5.7: The average optimality gap in the multiple-machine environment is compared to the genetic algorithm

Scenario name	re-entrant				no re-entrant			
	$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$		$\frac{z-z^{PB}}{z^{PB}}$		$\frac{z-z^{TB}}{z^{TB}}$	
	mean	SD	mean	SD	mean	SD	mean	SD
base case	0.1992	0.1305	2.9241	1.3093	0.0272	0.0253	3.2452	7.0354
no bottleneck	0.0941	0.0328	0.6766	0.3231	0.0363	0.0287	0.2144	0.1751
significant bottleneck	0.1912	0.0602	26.3715	77.5863	0.0330	0.0262	0.2638	0.4461
tight queue time limit	0.2417	0.1105	2.8983	1.1980	0.0314	0.0191	2.3886	1.9634
loose queue time limit	0.1996	0.1099	2.1221	0.8711	0.0275	0.0242	2.4312	3.6690
tight due time	0.1632	0.0864	1.2518	0.4374	0.0236	0.0195	0.6751	0.3618
loose due time	0.3898	0.1994	28.9644	28.2586	0.0485	0.0468	4.3113	14.2811
short maintenance length	0.3840	0.1095	2.8477	1.2927	0.0545	0.0394	0.0823	0.1242
long maintenance length	0.2455	0.0551	3.9953	1.4999	0.0205	0.0112	0.2374	0.3594
no weight	0.1988	0.0873	4.6632	2.6473	0.0286	0.0226	8.0112	5.3996
obvious weight discrepancy	0.2691	0.1512	1.9018	1.401	0.0460	0.0330	0.9454	0.9994
Average	0.2342	0.1030	7.147	10.6204	0.0343	0.0269	2.0733	3.1650

execution time of the greedy procedure is 0.0024 seconds, the average execution time of the genetic algorithm is 8.2867 seconds, and the average execution time of the MIP model is 2700.0354 seconds. This significant difference can also be observed in a no re-entrant environment. This advantage makes our algorithm more likely to be adopted in practical job and maintenance scheduling tasks compared to the MIP model.

doi:10.6342/NTU202400496

Table 5.8: The computation time (sec) in the multiple-machine environment

Scenario name	re-entrant			no re-entrant		
	Greedy	Genetic	MIP	Greedy	Genetic	MIP
base case	0.0031	8.3911	2700.0299	0.0018	8.4667	2526.3842
no bottleneck	0.0023	7.8061	2700.0372	0.0022	6.8876	1957.4416
significant bottleneck	0.0023	8.5035	2700.0373	0.0019	7.0985	1681.3967
tight queue time limit	0.0024	8.4158	2700.0364	0.0032	7.0976	2591.5367
loose queue time limit	0.0023	8.252	2700.0348	0.0019	7.0448	2302.5348
tight due time	0.0023	8.1969	2700.0336	0.0022	7.0413	2531.4843
loose due time	0.0023	8.2684	2700.0297	0.0022	7.0452	2297.215
short maintenance length	0.0024	8.3163	2700.0356	0.0024	7.1668	767.9364
long maintenance length	0.0024	8.4031	2700.0430	0.0019	7.0310	1692.2663
no weight	0.0024	8.2641	2700.0373	0.0031	7.1381	2700.0458
obvious weight discrepancy	0.0026	8.3367	2700.0350	0.0018	7.0729	2191.9483
Average	0.0024	8.2867	2700.0354	0.0022	7.1900	2112.7446

doi:10.6342/NTU202400496

Chapter 6

Case Study

6.1 Company overview

The problems addressed in this paper are derived from the actual situations of the companies we work with. Therefore, in this section, we validate the effectiveness of our proposed algorithms in the real world by applying our designed algorithms to actual company data. First, we provide an overview of our collaborative company background.

The company we cooperate with is a prominent Taiwanese electronics company in Taiwan and is well-known for its production of panel displays. The most important product for both the market and the company is TFT-LCD, and this product is also the product targeted by the content of our cooperation. The TFT-LCD production process consists of three main parts: array, cell, and module assembly. Our collaboration focuses specifically on the array production process, which involves the Thin Film deposition stage, the Photolithography stage, and the Etching stage. It is important to note that

doi:10.6342/NTU202400496

the Photolithography stage is identified as a bottleneck in the array process. Therefore, our algorithm aims to address the job scheduling problem by considering preventive maintenance in the bottleneck scenario, and the bottleneck is designed to be at the mid-process station.

In many companies, including ours, production and maintenance are typically managed by separate teams, which are the production team and the engineering team, each with its own key performance indicators (KPIs). For the production team, the goal is to meet demand and ensure timely production by arranging a suitable production schedule. On the other hand, the engineering team focuses on maintenance to maintain output quality and efficiency; moreover, limited queue time for work-in-progress (WIP) also poses challenges to scheduling maintenance activities between stages. Therefore, the unsynchronized decision is not the appropriate method in this situation, for example, preventive maintenance may occupy the machine's production resources, which may result in delays in production or the need to rework products due to exceeding the queue time limit.

In order for the team to reach a consensus and address the above challenges, our algorithms may serve as a reference and allow the goals of both teams to be achieved under reasonable circumstances.

6.2 Data description and parameter estimation

In the case study, due to the impact of the Corona Virus Disease 2019 (COVID-19) on the economy, among the lot data we obtained, the production output in February was

doi:10.6342/NTU202400496

the highest. Therefore, we use historical lot data from February 10, 2022, to February 28, 2022, provided by the collaborating company, where each day represents one instance. Here lot represents the set of sheets that each job needs to process in each process step.

These lot data contain the data of the lot that needs to be manufactured in the three stages: Thin Film deposition, Photolithography, and Etchingz, and also include processing time, number of sheets, the product type of job, queue time limit of the lot, and many other columns. Table 6.1 shows the sample of lot data. Each job consists of 24 production steps that need to be completed, and it is essential to adhere to the queue time limit for lots between each step, as depicted in Figures 3.1 and 3.2.

Table 6.1: Sample data of lot data

Data Generation Time	Product Code	Lot ID	Machine ID	Process ID	Queue Time Limit	...
2022/02/09 08:06:00	P01	L001	PHA0	PHP1	1440	...
2022/02/09 08:06:00	P02	L002	PHB0	PHP4	900	...

Moreover, In order to measure the severity of the job tardiness, we use the cost of backorders of each product provided by the company as the penalty weight of the job tardiness. Table 6.2 shows the sample data of the cost of backorders. It contains information about various costs, such as product code, item, due time, and unit cost.

For our problem scenario, these lot data lack some necessary detailed information, including job release time, job due time, and machine maintenance plan. Therefore, in this experiment, we estimate or configure these essential details using other information. First, we use the time when the lot data is generated as the job release time. However,

doi:10.6342/NTU202400496

Table 6.2: Sample data of the cost of backorders

Product Code	Item	Due time (h)	Unit Cost
P01	I01	12	0.2145
P02	I02	24	0.2145

since the data in February does not include the data from the first process, we estimate the time when the lot data is generated by averaging the data from April. Second, for the job due time, we use the average job completion time and job release time to estimate the due time of each job. Due to the aforementioned reason, the estimated data used here is also from April.

Finally, the distribution of the number of machines on the station is shown in Figure 6.1. For maintenance scheduling, the company has both monthly and weekly maintenance. Monthly maintenance requires 480 minutes, while weekly maintenance only requires 30 minutes. In this real-world scenario, most days are spent performing weekly maintenance instead of monthly maintenance. Therefore, our experiment only considers the more frequently occurring weekly maintenance. Moreover, in the case study experiment, besides the MIP Model, we also utilize the greedy procedure and the genetic algorithm. Regarding the experiments with the genetic algorithm, we set the algorithm to provide the scheduling result with the minimum total weighted tardiness among all considered schedules after 50 iterations. However, if the best result remains the same for 10 consecutive iterations, the algorithm will stop. Additionally, similar to our exper-

doi:10.6342/NTU202400496

iments in Chapter 5, we generate 20 initial schedules initially, and each offspring has a mutation probability of 0.01.

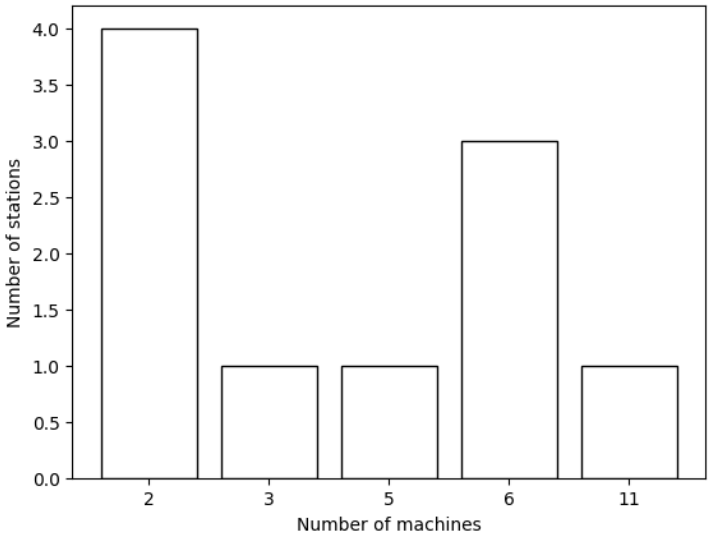


Figure 6.1: The distribution of machines on the station

We refer to the average daily machine maintenance ratio of our company, which is 8.89%. Each day, we randomly select 4 machines out of 45 machines for maintenance. These machines are then assigned weekly maintenance. The release time of maintenance is also when the lot data is generated. In order to test the performance of our algorithm, we also randomly select a number within the range of 0 to half of the number of machines on the station to determine the number of machines to be maintained at each station. These machines are then assigned weekly maintenance. The release time of maintenance is also when the lot data is generated. Table 6.3 contains the number of machines, number of required maintenance machines, and the percentage of required maintenance machines for each day. In Table 6.3, we refer to the proportion of machines requiring maintenance as *RMM*.

doi:10.6342/NTU202400496

Table 6.3: The setting of required maintenance machines for each day

Date	number of machines	number of RMM	percentage of RMM
2022/02/10	45	15	33.33%
2022/02/11	45	19	42.22%
2022/02/12	45	12	26.67%
2022/02/13	45	12	26.67%
2022/02/14	45	12	26.67%
2022/02/15	45	10	22.22%
2022/02/16	45	14	31.11%
2022/02/17	45	12	26.67%
2022/02/18	45	11	24.44%
2022/02/19	45	10	22.22%
2022/02/20	45	10	22.22%
2022/02/21	45	15	33.33%
2022/02/22	45	16	35.56%
2022/02/23	45	16	35.56%
2022/02/24	45	10	22.22%
2022/02/25	45	9	20.00%
2022/02/26	45	14	31.11%
2022/02/27	45	16	35.56%
2022/02/28	45	13	28.89%

doi:10.6342/NTU202400496

6.3 Experiment result

During the specified date range in the experimental data, the number of jobs to be scheduled ranges from 300 to 600 in each day, and each job consists of 24 steps; moreover, maintenance time for the machines needs to be scheduled. Both of the above situations result in a big-scale problem. When using the MIP Model to handle these cases, we encounter memory errors that prevent us from finding the optimal schedule for these real cases. However, our algorithms, including the greedy procedure and the genetic algorithm, still deliver satisfactory performance in addressing these real-world scenarios. The following two subsections show the experimental results of two different types of maintenance machine ratio settings.

6.3.1 Current maintenance-machine ratio

Figure 6.2 shows the total weighted tardiness calculated based on the schedules generated by two different algorithms, respectively, from February 10, 2022, to February 28, 2022. Four machines are chosen randomly each day for maintenance. The average objective value of the scheduling obtained by the greedy procedure is 16036.91. The average objective value of the genetic algorithm is 10119.27 for the obtained scheduling.

From the number of tardy jobs, we can demonstrate the performance of our algorithm more clearly. Table 6.4 contains the experiment results of the greedy procedure and the genetic algorithm, including the number of tardy jobs, the percentage of job tardy, and the number of jobs. From Table 6.4, it can be seen that on average, out of 494 jobs, the greedy procedure only causes a tardy in 33 jobs, with a tardy percentage of 6.66%.

doi:10.6342/NTU202400496

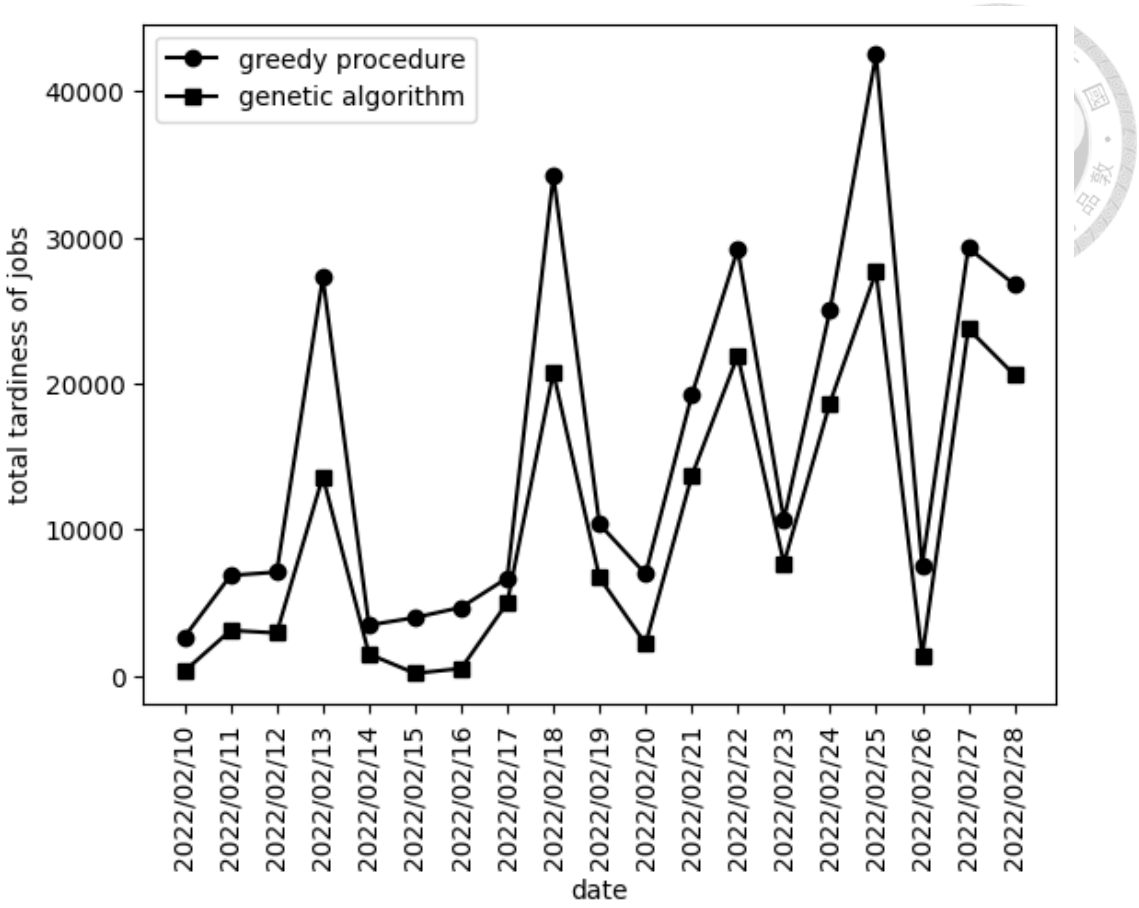


Figure 6.2: The objective value of the case study experiment on the current maintenance-machine ratio setting

On the other hand, the genetic algorithm only causes a tardy in 27 jobs, with a tardy percentage of 5.33%.

In addition to the aforementioned results, the execution time of our algorithm is also one of the advantages of our algorithm. Figure 6.3 shows the time required by our algorithm for different dates. On average, the greedy procedure can complete the scheduling job within 0.29 minutes, while the genetic algorithm takes about 10 minutes to complete the scheduling job.

doi:10.6342/NTU202400496

Table 6.4: The experiment result of the case study on the current maintenance-machine ratio setting

Date	number of jobs	greedy procedure		genetic algorithm	
		number of tardy jobs	tardy-persentage	number of tardy jobs	tardy-persentage
2022/02/10	380	9	2.37%	3	0.79%
2022/02/11	399	24	6.02%	20	5.01%
2022/02/12	402	24	5.97%	21	5.22%
2022/02/13	421	46	10.93%	36	8.55%
2022/02/14	456	11	2.41%	7	1.54%
2022/02/15	476	12	2.52%	3	0.63%
2022/02/16	478	13	2.72%	4	0.84%
2022/02/17	502	27	5.38%	24	4.78%
2022/02/18	528	55	10.42%	48	9.09%
2022/02/19	504	24	4.76%	19	3.77%
2022/02/20	486	15	3.09%	9	1.85%
2022/02/21	509	25	4.91%	23	4.52%
2022/02/22	529	54	10.21%	43	8.13%
2022/02/23	593	25	4.22%	24	4.05%
2022/02/24	533	53	9.94%	49	9.19%
2022/02/25	539	74	13.73%	62	11.5%
2022/02/26	588	28	4.76%	12	2.04%
2022/02/27	536	61	11.38%	55	10.26%
2022/02/28	531	57	10.73%	51	9.6%
Average	494	33	6.66%	27	5.33%

6.3.2 Higher maintenance-machine ratio

Figure 6.4 shows the total weighted tardiness calculated based on the schedules generated by two different algorithms. Every day, 10 to 20 machines are randomly selected for

doi:10.6342/NTU202400496

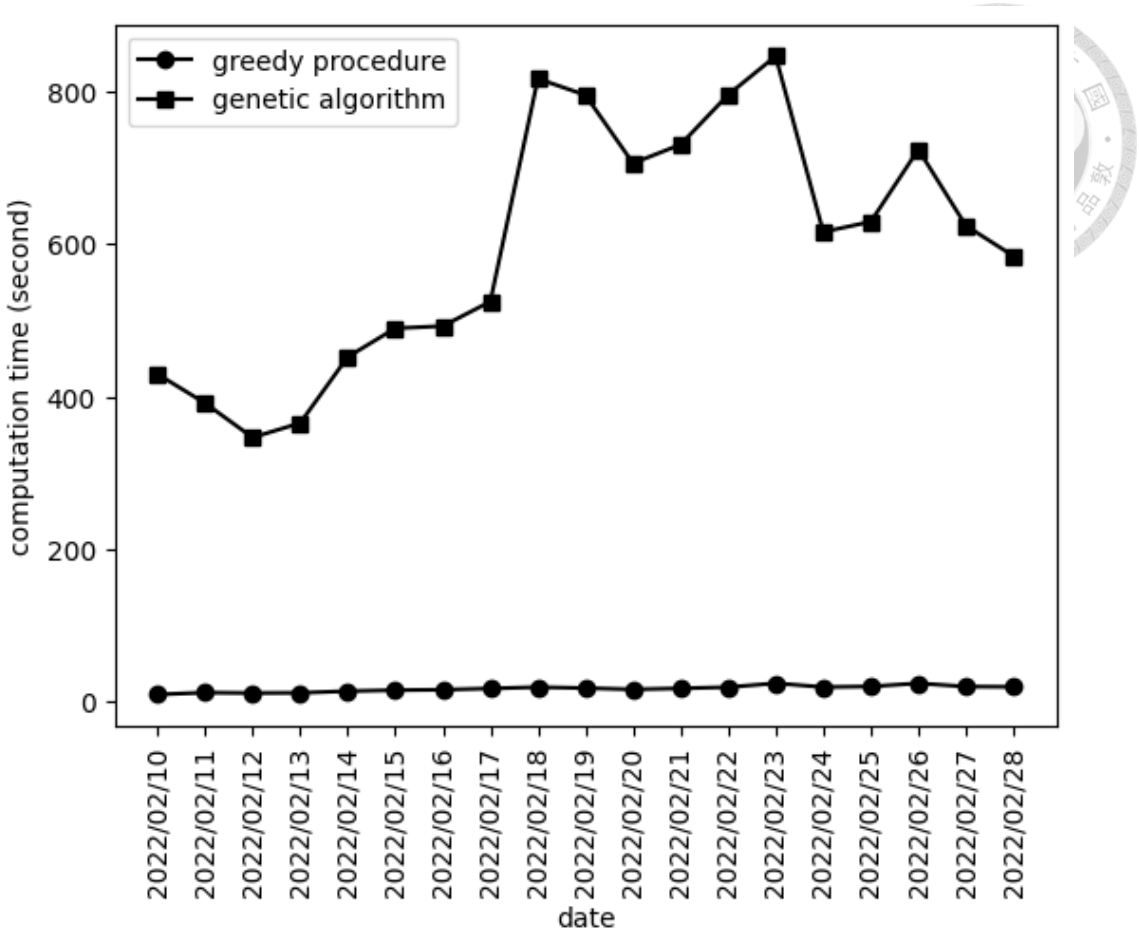


Figure 6.3: The computation time (second) of the case study experiment on the current maintenance-machine ratio setting

maintenance from a pool of 45 machines. The average objective value of the scheduling obtained by the greedy procedure is 14360.16. The average objective value of the genetic algorithm is 9988.67 for the obtained scheduling.

Table 6.5 contains the experiment results of the greedy procedure and the genetic algorithm, including the number of tardy jobs, the percentage of job tardy, and the number of jobs. From Table 6.5, it can be seen that on average, out of 494 jobs, the greedy procedure only causes a tardy in 32 jobs, with a tardy percentage of 6.31%. On the other hand, the genetic algorithm only causes a tardy in 26 jobs, with a tardy

doi:10.6342/NTU202400496

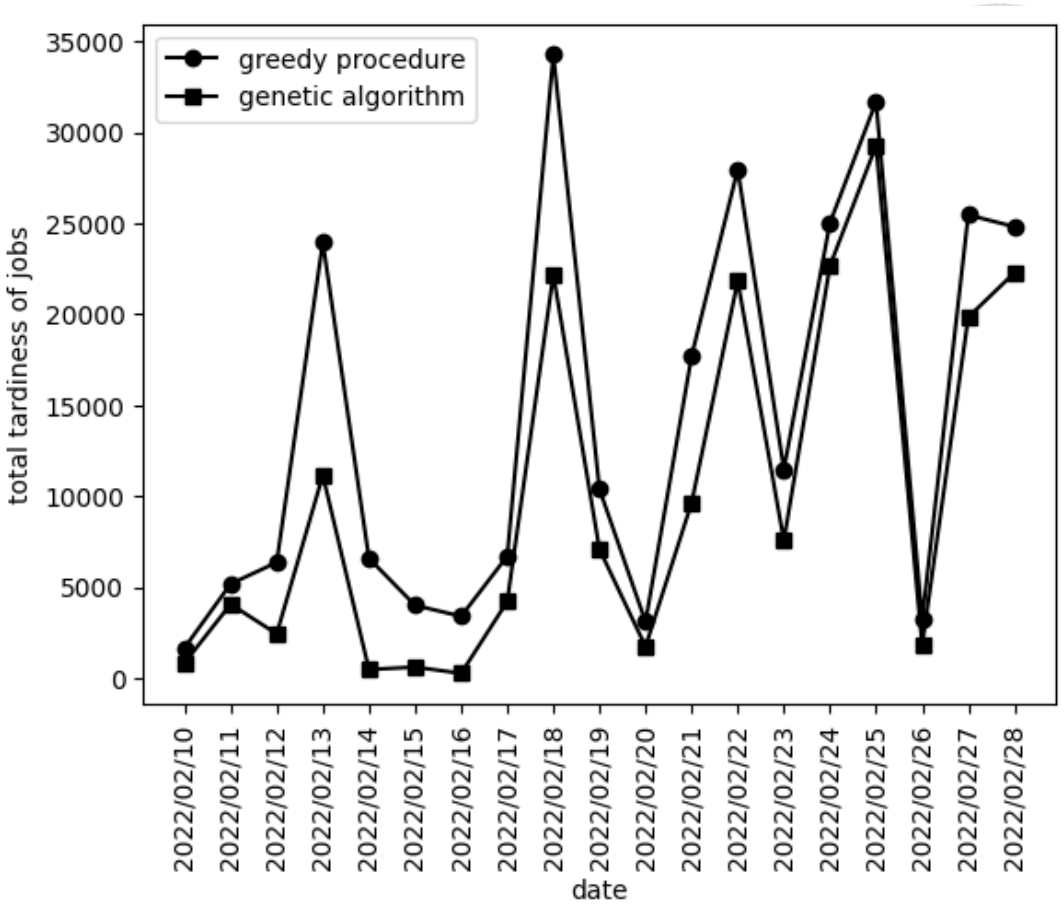


Figure 6.4: The objective value of the case study experiment on the higher maintenance-machine ratio setting

percentage of 5.16%.

Figure 6.5 shows the execution time of our algorithm for different dates. On average, the greedy procedure can complete the scheduling job within 0.21 minutes, while the genetic algorithm takes about 9.6 minutes to complete the scheduling job.

doi:10.6342/NTU202400496

Table 6.5: The experiment result of the case study on the higher maintenance-machine ratio setting

Date	number of jobs	greedy procedure		genetic algorithm	
		number of tardy jobs	tardy-persentage	number of tardy jobs	tardy-persentage
2022/02/10	380	7	1.84%	5	1.32%
2022/02/11	399	18	4.51%	15	3.76%
2022/02/12	402	25	6.22%	17	4.23%
2022/02/13	421	39	9.26%	35	8.31%
2022/02/14	456	15	3.29%	4	0.88%
2022/02/15	476	12	2.52%	5	1.05%
2022/02/16	478	11	2.30%	3	0.63%
2022/02/17	502	27	5.38%	22	4.38%
2022/02/18	528	55	10.42%	49	9.28%
2022/02/19	504	24	4.76%	22	4.37%
2022/02/20	486	11	2.26%	8	1.65%
2022/02/21	509	25	4.91%	21	4.13%
2022/02/22	529	50	9.45%	42	7.94%
2022/02/23	593	27	4.55%	16	2.70%
2022/02/24	533	57	10.69%	55	10.32%
2022/02/25	539	62	11.50%	60	11.13%
2022/02/26	588	18	3.06%	14	2.38%
2022/02/27	536	59	11.01%	52	9.70%
2022/02/28	531	64	12.05%	52	9.79%
Average	494	32	6.31%	26	5.16%

doi:10.6342/NTU202400496

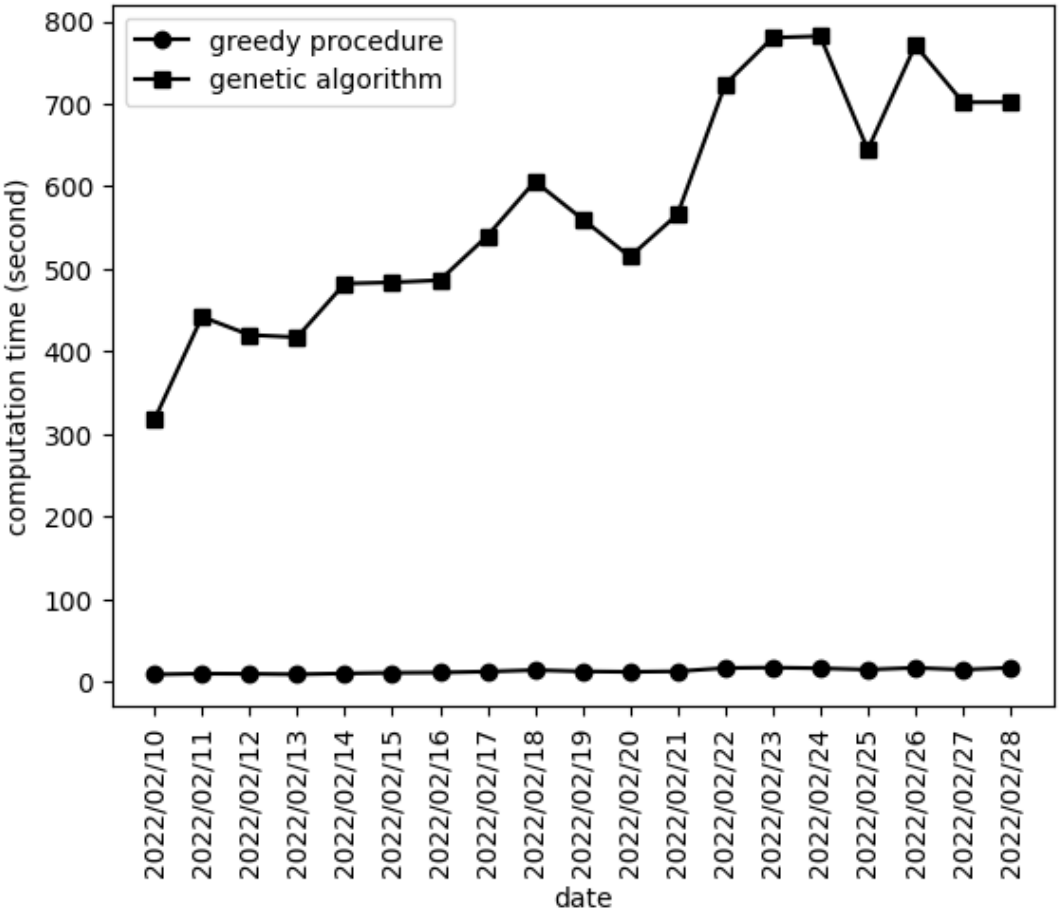


Figure 6.5: The computation time (second) of the case study experiment on the higher maintenance-machine ratio setting

doi:10.6342/NTU202400496

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

This study examines a joint production scheduling and preventive maintenance problem in a job shop environment. Since there are already many methods available to assess machine conditions, we assume prior knowledge of machines requiring maintenance. Moreover, we also consider the queue time limit condition and design the algorithm while considering the bottleneck station. Therefore, our algorithm can better meet the needs of the production end in practice.

The main purpose of this study is to find an efficient method for scheduling jobs and maintenance, with the goal of minimizing total weight tardiness. We formulate the problem into a mixed integer linear problem, and we use the objective value of the MIP model as the lower bound to evaluate the performance of our algorithm. However, because this is an NP-hard problem, the MIP model cannot obtain the optimal solution or

doi:10.6342/NTU202400496

even a feasible solution within a reasonable time frame for large-scale problem instances. Therefore, in this study, we propose a greedy procedure consisting of three parts: job listing, maintenance scheduling, and job scheduling. The most notable aspect is in the job scheduling part, where we employ a backward checking approach to prioritize tasks while ensuring compliance with the queue time limit constraint. Additionally, we enhance the performance of our algorithm by utilizing a genetic algorithm to explore different job processing orders.

Under our problem setting, the greedy procedure and genetic algorithm perform almost equally. Whether in a single-machine or multi-machine environment, the performance is better in no re-entrant environment. Compared to the primal lower bound, in the case of a single-machine environment without re-entrant, the average objective value is 0.0819 with a standard deviation of 0.0619. Among these scenarios, the significant bottleneck scenario performs the best. In the case of a multi-machine environment with the no re-entrant environment, the average objective value is 0.0346 with a standard deviation of 0.0269. Among these scenarios, the long maintenance length scenario and tight due time scenario perform the best.

Finally, we used an algorithm to schedule the actual job data of a Taiwanese panel production company while considering a pre-determined machine maintenance plan. Based on the experimental results from February's data, the greedy procedure, which takes into account conditions such as queue time limits, overlapping maintenance of the same station, completing maintenance within a certain timeframe, and so on, only resulted in a delay for an average of 6.31% of jobs under the current maintenance-machine scenario and 6.31% of jobs under the higher ratio scenario. The genetic algorithm performed even

doi:10.6342/NTU202400496

better, with an average of only 5.33% and 5.16% of jobs tardy. Additionally, the average execution times for the two algorithms were 0.21 minutes and 9.6 minutes, respectively. These findings demonstrate the high practicality of our proposed algorithm in an actual panel production factory, as it can quickly provide suitable job and maintenance schedules.

7.2 Future directions

From our experimental results, we can observe that our algorithm performs poorly in the re-entrant environment. This is because, according to the logic of our algorithm, tasks that return to the same station repeatedly result in idle time on the machine, leading to increased job tardiness. Although our algorithm performs well in the no re-entrant environment, it is still worth exploring how to address the challenges encountered in the re-entrant environment.

Furthermore, our current design of the genetic algorithm performs almost the same as the greedy procedure in the various scenarios we have considered. This could be because the sorting rule of WEDD takes both job due time and penalty into account, which is directly related to our objective. Therefore, it suits our problem well. However, it is also possible that the design of the genetic algorithm is not complicated enough, resulting in limited consideration of job processing orders. Therefore, optimizing the genetic algorithm to improve its effectiveness is an area that can be explored in the future.

doi:10.6342/NTU202400496

Bibliography

Abdul-Razaq, T.S., C.N. Potts, L.N. Van Wassenhove. 1990. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics* **26**(2) 235–253.

Akkerman, Renzo, Dirk Pieter Van Donk, Gerard Gaalman. 2007. Influence of capacity- and time-constrained intermediate storage in two-stage food production systems. *International Journal of Production Research* **45** 2955–2973.

Allahverdi, A., H. Aydilek. 2015. The two stage assembly flowshop scheduling problem to minimize total tardiness. *Journal of Intelligent Manufacturing* **26** 225–237.

Biskup, D., J. Herrmann, J. N.D. Gupta. 2008. Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics* **115**(1) 134–142.

Chen, C.L., T.I.n Tang. 2012. Flexible flow line scheduling problems with re-entrant flows and queue-time constraints. *IET Conference Publications* **2012** 1065–1068.

Chen, J.-F., T.-H. Wu. 2006. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. *Omega* **34**(1) 81–89.

doi:10.6342/NTU202400496

Choi, H.-S., D.-H. Lee. 2009. Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops. *Computers & Industrial Engineering* **56**(1) 113–120.

de Alba, H. G., S. Nucamendi-Guillén, O. Avalos-Rosales. 2022. A mixed integer formulation and an efficient metaheuristic for the unrelated parallel machine scheduling problem: Total tardiness minimization. *EURO Journal on Computational Optimization* **10** 100034.

Du, J., J.Y.-T. Leung. 1990. Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research* **15**(3) 483–495.

Kanet, J.J., X. Li. 2004. A weighted modified due date rule for sequencing to minimize weighted tardiness. *Journal of Scheduling* **7** 261–276.

Liao, L.-M., C.-J. Huang. 2010. Tabu search for non-permutation flowshop scheduling problem with minimizing total tardiness. *Applied Mathematics and Computation* **217**(2) 557–567.

Liaw, C.-F., Y.-K. Lin, C.-Y. Cheng, M. Chen. 2003. Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers & Operations Research* **30**(12) 1777–1789.

Su, Ling-Huey. 2003. A hybrid two-stage flowshop with limited waiting time constraints. *Computers & Industrial Engineering* **44** 409–424.

Vallada, E., R. Ruiz, G. Minella. 2008. Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research* **35**(4) 1350–1373.

doi:10.6342/NTU202400496

Wang, H. 2002. A survey of maintenance policies of deteriorating systems. *European Journal of Operational Research* **139**(3) 469–489.

Yalaoui, F., C. Chu. 2002. Parallel machine scheduling to minimize total tardiness. *International Journal of Production Economics* **76**(3) 265–279.

Yang, Dar-Li, Maw-Sheng Chern. 1995. A two-machine flowshop sequencing problem with limited waiting time constraints. *Computers & Industrial Engineering* **28** 63–70.

Yu, J.-M., R. Huang, D.-H. Lee. 2017. Iterative algorithms for batching and scheduling to minimise the total job tardiness in two-stage hybrid flow shops. *International Journal of Production Research* **55**(11) 3266–3282.

doi:10.6342/NTU202400496